(031) 9002

NH99'ers User Group

New Hampshire 99'ers User Group
PO Box 5991, Manchester, NH 03108

Newsletter

February 1990
Vol. 8, No. 2

CLUB NEWS

by Paul Bendeck, President

Well, it's been snowing now for 2 days and I'm tired of shoveling the snow so I thought I would do something a little more relaxing like work on this article for the newsletter. Besides all the snow, 1990 brought renewed interest in both the club and the TI-99/4A. The club started the new decade with a very entertaining and interesting meeting in January. I'll try to cover some of the highlights.

Thanks to Elliot Hardy, the club has expanded our assets with a complete set of 99'er Magazines plus all of the disk software that was distributed with the magazines. In case you have never seen this magazine (they stopped publishing about 3-4 years ago), this was the best all around magazine covering the TI-99/4A computer. Each issue is filled with very descriptive articles on every subject about the TI-99/4A. Each issue also included several excellent programs which were distributed on disk. These programs were not only fun to play with but also educational because they illustrated various programming techniques. Each of these programs was fully documented and described in the magazine. The magazines are on file with the club and are available for checking out by club members. See Phil Davis the newsletter librarian for details. All of the disk software is now in the club software library. Copies can be obtained at the meeting.

Speaking of the software library, things are looking up. Besides the 99'er Magazine software, there were several other donations from club members. Mike Scanlon demonstrated a disk of very creative games he picked up from the NUTMEG User's Group. The single disk included 4 games: Backgammon, Solitaire, Monopoly, and Witch/Hour, plus a menu loader. All of these games ran in Extended Basic and had lots of good graphics. This disk is now available in the software library.

Curtis Provance brought in a copy of one of the best Chess programs I have seen for the TI-99/4A. It's called Sargon Chess and includes programmable levels of difficulty and very good graphics. This program uses standard Chess notation for making moves which makes it easy to learn and use. Sargon Chess is available now in the software library.

Probably the best demo of the evening was provided by Vince Demers, our newsletter editor. Vince showed us a game called Carfax Abbey. This is a maze game consisting of a castle with a lot of rooms (25 rooms per floor, and a total of 4 floors). Each room has multiple exits and secret passageways to other rooms, sometimes on different floors. The graphics were top notch in this game. If you want to find out what is in these rooms, you'll have to buy the disk from our software library.

As a reminder, software disks are available from the software library for a fee of $3.50. Copies can be made during regularly scheduled club meetings. Chris Agrafiotis, our software librarian, has promised to bring an updated copy of the library catalog to the next meeting. (Cont. - page 7)

DISK DOCTOR

Curtis Alan Provance
Paragon Computing

One member brought a 'damaged' floppy to the last meeting. Something had happened to the disk and all his files were gone! Actually, he knew that the information was still there - but he had no access to it. Within a few minutes, we had successfully recovered all the files ....

This is the first in a series of articles on how your disk controller works, and how you can recover from several common disk errors. In order to examine and repair your own disk, you will need a disk utility program which allows you to read and write individual disk sectors. John Birdwell's excellent FAIRWARE offering, "Disk Utility" is what I will be using. You may obtain the complete program along with disk based documentation by sending a minimum donation of $10 to:
John Birdwell
7052 Springhill Circle
Eden Praire, MN 55344

Secondly, I will be using a 'training' disk and refering to specific sectors - and bytes within those sectors. To ensure that your 'training' disk looks like mine, you should format (or reformat) a disk and name it "BLANK". Please format it as a Single-Sided, Single-Density disk (SS/SD) - and don't write-protect it!

Finally, if you are not familiar with using hexidecimal notation, please stop and read the article on hexidecimal found elsewhere in this newsletter. The bulk of this discussion will use hexidecimal (Disk Utilities 'native' mode). Hexidecimal is not hard to learn, and the few minutes you spend on the other article will save hours of frustration later when you do your own disk 'doctoring.'

TERMINOLOGY

Let's review some terminology (as it applies specifically to TI99/4A computers):

SECTOR: A continuous section of your disk which holds 256 bytes of information. Each byte can have any one of 256 values. Other computers may have 256 byte sectors, 512 byte sectors, or whatever. There is information stored in between the sectors, but we won't concern ourselves with it during the course of these articles.
TRACK: A continuous collection of sectors. TI99/4A disks typically have 9 sectors (Single-Density) or 18 sectors (Double-Density) in each track (although MYARC's preferred number of sectors for Double Density disks is 16). A TI99/4A floppy usually has 40 tracks per side (some have only 35 - more on that later).
SIDES: How many sides of a disk are used, either a Single-Sided (usually the side without the label) or Double-Sided disk.
DENSITY: How many sectors are packed in each track - typically 9 for Single-Density and 18 (or 16) for Double-Density.

Therefore, if you do your math right, a Single-Sided, Single-Density (SS/SD) disk having 256 bytes per sector, 9 sectors per track, and 40 tracks per side has the capability of holding 360 sectors, or 92160 bytes (usually just called "90K" because a "K" is 1024 bytes). A Double-Sided, Single-Density (DS/SD) disk (or Single-Sided, Double-Density) (SS/DD) holds 180K, and a Double-Sided, Double-Density (DS/DD) disk holds 360K (all figures based on 9 or 18 sectors per track, as appropriate).

DISK USAGE

Computer makers use different formats for their floppy disks. This includes how the tracks and sectors are set up, as well as which sectors are used for control information. 'Soft-sectored' disks can be formatted in a variety of ways - TI's way, IBM's way, or whatever. We will only deal with TI's format during these articles.

TI chose to use the first two disk sectors for their control information. This is why you get the message "358 sectors free" when you format a good SS/SD disk. There are really 360 sectors - but two are needed by the controller.

SECTOR 0

The first sector on a disk (all the way out on the rim) is sector 0. Let's see what the first sector looks like. Load the disk utility and select "SECTOR EDIT" from the menu. Place the disk in drive 1, select drive 1, then select sector 0 as the sector you want to edit. The screen should look like this:

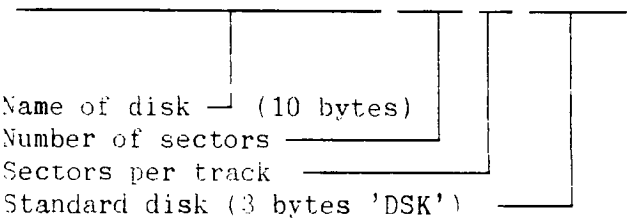
```
424D 414F 4C20 2020 2020 0168 0944 534B
2028 0101 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0300 0000 0000 0000
... (the rest are all 0000's or FFFF's)
```

If you want to see the characters in ASCII, press the control and A keys at the same time. The screen should now be:

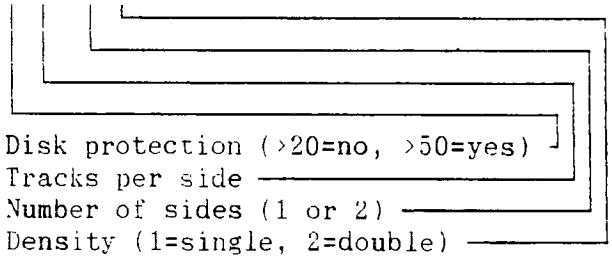
```
B L  A N  K              . h  . D  S K
  (  .  .  .  .  .  .  .  .  .  .  .  .
...(the rest all appear as . .)
```

This sector holds the following data:

```
B L  A N  K              . h  . D  S K
424D 414F 4C20 2020 2020 0168 0944 534B
```




```
.  .  .  .  .  .  .  .  .  .  .  .  .  .
2028 0101 0000 0000 0000 0000 0000 0000
```



The remaining bytes on this line are reserved for future use.

Please note that the disk protection used here does not refer to the write protection you get by covering the notch on the side of the disk. This is a software protection scheme developed by TI which we will discuss in a moment.

The next row of bytes are also reserved for future use, along with the first eight bytes of the fourth row.

The ninth byte of the fourth row is the first byte of the sector bit-map. A bit-map means that each bit in each byte represents a larger object (in this case, a sector).

Each byte consists of eight bits, and each bit represents a sector. If a sector is currently being used (or if it is damaged and the controller couldn't initialize it) the bit will be set to 1. Otherwise, the bit is 0. Since sectors 0 and 1 are always in use, the first byte will always have the first two bits set.

It is important to note that bit 0 is the right-most bit. In other words, if you were to expand a byte into its individual eight bits, the bits would look like this: 7 6 5 4 3 2 1 0.

On a fresh disk the bit-map table will start with >0300 >0000 .... In other words, since >03 = 00000011, this bit map indicates that sectors 0 and 1 are in use. This bit-map can hold data for a DS/DD disk (takes 128 bytes). Since we formatted the disk as SS/SD, the last 96 bytes are FFFF - meaning that they aren't available.

DISK PROTECTION

The Disk Manager module suppplied with the TI disk controller contained an undocumented feature which would provide disk protection. This feature was enabled by pressing (I believe) the Control and X keys simultaneously - ten times. Once done, you could create a disk which was "protected" from being duplicated by the disk manager.

The only protection this feature provided was to alter the first byte in the second row from a space character to a 'P' (for protected) character. Until the advent of DM-1000, Disk Utilities, and similar programs, this protection thwarted most of the would-be copiers. It is useless now, except against users of TI's original Disk Manager cartridge.

NEXT MONTH: FILE HEADERS AND SECTOR 1

TYPE IN PROGRAMS

VICIOUS

This was taken from the West Penn NL Aug 1989. No credit given to an author. An XB program that will run in a bare console W/XBASIC. Will remind some of "SPYS". Very good and accepts either keyboard or joy-sticks. May download from Spirit of 99 BBS if you do not want to type it in.

```
100 @=1 :: _=2 :: CALL CLEAR
 :: CALL SCREEN(_):: RANDOMI
ZE :: CALL MAGNIFY(3):: FOR
A=@ TO 14 :: CALL COLOR(A,16
,@):: NEXT A :: CALL COLOR(1
1,11,@):: CALL CHAR(48,"007C
444444444447C")
110 CALL CHAR(96,"070B132141
81E3FFFFE3814121130B07E0D0C8
848281C7FFFFC7818284C8D0E")
120 CALL CHAR(100,"030C30204
0408080808040402030 0C03C0300
C040202010101010202040C30C",
108,"007E7E7E7E7E7E001C2A497
F492A1C")
130 CALL CHAR(112,"808080808
0808080010101010101010FF000
000000000000000000000000FF"
):: DISPLAY AT(@,8):"VICIOUS
 CIRCLE"
140 DISPLAY AT(4,@):"AVOID T
HE CIRCLES WHILE":"CLEARING
THE GRID.": :"USE THE JOYSTI
CK OR ARROW  KEYS TO MOVE."
150 DISPLAY AT(10,@):"YOU RE
CEIVE 10 PTS FOR EACH SQUARE
, OR 1000 PTS FOR AN  ENTIRE
 GRID.": :"ONCE YOU HAVE BEE
N HIT 10  TIMES, THE GAME W
ILL END."
160 DISPLAY AT(17,@):"FOR EV
ERY 5000 PTS, YOU":"GET AN E
XTRA LIFE." :: DISPLAY AT(23
,7):"JOYSTICKS?(Y/N)"
170 CALL KEY(I,B,C):: IF C=I
 THEN 170
180 D=I :: IF B=89 OR B=121
THEN D=@ :: IF B=121 THEN 21
0
190 IF D=I THEN 210
200 FOR A=@ TO 4 :: DISPLAY
AT(23,4):"RELEASE ALPHA-LOCK
 KEY" :: FOR E=@ TO 40 :: NE
XT E :: DISPLAY AT(23,4):: N
EXT A
210 CALL CLEAR
220 F=85 :: G=117 :: H,I,J=I
 :: K=10 :: L=5000 :: CALL H
CHAR(_,9,115,17):: CALL HCHA
R(20,9,114,17):: CALL VCHAR(
3,8,113,17):: CALL VCHAR(3,2
6,112,17)
230 DISPLAY AT(@,_):"00000"
240 FOR A=4 TO 18 STEP _ ::
DISPLAY AT(A,8)SIZE(16):RPT$
("1 ",8):: NEXT A :: FOR A=_
 TO 9 :: IF A/_=INT(A/_)THEN
 M=-@ ELSE M=@
250 M=M*INT(RND*10+12+J):: C
ALL SPRITE(#A,100,INT(RND*14
+3),200,A*16+37,M,I):: NEXT
A :: GOSUB 420
260 DISPLAY AT(22,9):"PRESS
ANY KEY" :: DISPLAY AT(22,9)
:: CALL KEY(I,N,O):: CALL KE
Y(@,P,Q):: IF O=I AND Q=I TH
EN 260 ELSE CALL SOUND(500,2
62,3,330,3,392,3)
270 CALL SPRITE(#@,96,15,F,G
):: GOTO 300
280 IF D=I THEN 390 ELSE CAL
L JOYST(@,B,C):: IF ABS(B)=A
BS(C)THEN 320
290 G=MIN(181,MAX(69,G+B*4))
:: F=MIN(133,MAX(21,F-C*4)):
: CALL LOCATE(#@,F,G)
300 CALL GCHAR(INT(F/8)+_,IN
T(G/8+_),X):: IF R<>X THEN
 320 ELSE CALL SOUND(140,-6,
3,900,4,1100,5,1300,6)
310 CALL HCHAR(INT(F/8+_),IN
T(G/8+@),32,_):: I=I+@ :: IF
 I=64 THEN 330
320 CALL COINC(ALL,R):: IF R
=I THEN 280 ELSE CALL SOUND(
200,-6,_):: K=K-@ :: GOSUB 4
20 :: IF K=I THEN 350 ELSE 2
80
330 CALL SOUND(1600,131,_,39
2,_,1047,_):: J=J+_ :: CALL
DELSPRITE(ALL):: I=I+@ :: F=85
 :: G=117 :: H=H+1000 :: DIS
PLAY AT(@,7-LEN(STR$(H)))SIZ
E(6):STR$(H)
340 IF H=L THEN K=K+@ :: GOS
UB 420 :: L=L+500 :: GOTO 24
0 ELSE 240
350 FOR A=@ TO I :: H=H+10 :
: CALL SOUND(30,523,_):: DIS
PLAY AT(@,7-LEN(STR$(H))):ST
R$(H):: CALL SOUND(20,200,30
):: NEXT A :: DISPLAY AT(22,
11):"GAME  OVER" :: FOR A=@
TO 340 :: NEXT A
360 DISPLAY AT(22,8):"PLAY A
GAIN?(Y/N)"
370 CALL KEY(I,B,C):: IF C=I
 THEN 370
380 IF B=89 OR B=121 THEN CA
LL DELSPRITE(ALL):: CALL CLE
AR :: GOTO 220 ELSE END
390 CALL KEY(I,N,O):: B,C=I
:: IF N=83 OR N=115 THEN B=-
4 ELSE IF N=68 OR N=100 THEN
 B=4
400 IF N=69 OR N=101 THEN C=
4 ELSE IF N=88 OR N=120 THEN
 C=-4
410 GOTO 290
420 DISPLAY AT(@,16):RPT$("
",13-K)&RPT$("m",K):: RETURN
```

STRESS SYNDROME

This article and program was taken from the PUNN newsletter - issue Aug 1989 STRESS is on the SPIRIT OF 99 BBS if you do not want to type it in.

This month we are offering a little program to test your courage, patience and composture. It is very easy to type in, just be sure you check the DATA numbers carefully before you run the program.

I have heard that some folks have had a severe stress syndrome after running this program, but I am sure that none of our members have any of those symptoms.

However, run the program at your own risk and the Editor and the entire PUNN staff will not be responsible in any way for liabilities.

Chuck Ball, Editor

```
100 REM SAVE DSK1.HELLO
110 REM
120 REM Mystery Program
130 REM by Chris Schan
140 REM
150 REM Requires Memory Expa
nsion
160 REM and Synthesizer
170 REM
180 REM Runs in Extended Bas
ic or Console Basic
190 REM with Editor/Assemble
y or Mini-Memory
200 REM
210 REM
220 REM
230 DATA 71,64,72,65,70,75
240 DATA 73,70,76,67,66,66
250 DATA 65,68,76,68,77,68
260 DATA 78,71,77,66,68,66
270 DATA 66,67,74,67,74,77
280 DATA 74,68,73,71,64,67
290 DATA 72,68,76,65,72,68
300 DATA 76,65
310 CALL INIT
320 CALL PEEK(-28672,A)
330 IF A<>96 THEN 430
340 FOR Z=1 TO 11
350 FOR X=1 TO 4
360 READ A
370 CALL LOAD(-27648,A)
380 NEXT X
390 CALL LOAD(-27648,64)
400 CALL LOAD(-27648,80)
410 NEXT Z
420 STOP
430 PRINT "You don't have a
Speech"
440 PRINT "Synthesizer attac
hed!"
```

HEXIDECIMAL

Curtis Alan Provance
Paragon Computing

With a few exceptions, humans around the world have adopted the base ten numbering system. One can presume this is due to the ten digits found on our hands. While base ten is 'easy' for us, it is not always the best numbering system for the job. For computers, the best numbering system is binary. For computer programmers, the best numbering system is (usually) hexidecimal. Why use hexidecimal? What is a numbering system, anyway? Let's start with the second question first....

What is a numbering system, anyway?

You can think of the 'base' of a numbering system as the number of digits used to make numerals. My dictionary says (as one of many definitions): "The number that is raised to various powers to generate the principal counting units of a number system." In other words, if you write the numeral "2539", what you are saying is:
"2x1000 + 5x100 + 3x10 + 9x1"
I realize that I won't get a noble prize for this, but bear with me, please. We can take the about expression, and convert the 1000, 100, 10, and 1 into 'powers' of ten:
"$2x10^3 + 5x10^2 + 3x10^1 + 9x10^0$"
Please note that the 'powers' of ten correspond to the column number, assuming that we start at the right side with column zero. So, in base ten, we represent numbers by breaking the number into multiples of powers of ten.

Before introducing hexidecimal or binary, let's get a few terms straight. In particular, the terms digit, numeral, and number should be made clear.

DIGIT: "... 3a. Any one of the ten Arabic number symbols, 0 through 9. b. Such a symbol used in a system of numeration."

NUMBER: "A member of the set of positive integers; ...."

NUMERAL: "A symbol or mark used to represent a number."

Your dictionary may disagree, but let's go with these definitions for the moment. As you can see, digits are individual symbols used to make numerals. Numerals are symbols (made of one or more digits) which represent numbers. For example, FE, 376, 254, and 11111110 are all numerals representing the the number two hundred fifty four in base 16, base 8, base 10, and base 2, respectively. The first numeral is composed of two digits, the second and third numerals are composed of three digits, and the last - eight digits. As you may have guessed by now, there are the same number of digits as the base. Base ten (decimal) has ten digits (0 through 9), base 2 (binary) has two digits (0 and 1), base eight (octal) has eight digits (0 through 7) and base 16 (hexidecimal) has sixteen digits - whoa! After we use up 0 through 9, where do we get the other six digits?

I suppose the first people to use hexidecimal could have come up with weird looking characters for the last six digits, but they didn't. The last six digits are represented by the characters A through F (or a through f - if your program doesn't care about case). Therefore, in the example above, if we take the numeral FE in hexidecimal (base 16), we can write it as: "$Fx10_{16}^1 + Ex10_{16}^0$" - note that "$10_{16}$" in hexidecimal is the same as "16_{10}" in base ten. In other words, to represent "ten" in base ten, you write "10_{10}" - to represent "two" in base two, you write "10_2" - to represent eight in base eight, you write "10_8" - to represent "sixteen" in base sixteen, you write "10_{16}".

When we write a number in base ten, we often don't add the base value, i.e. we write 254 and not 254_{10}. When we want to write a number in another base, we need some way to indicate that it isn't in base ten. If we didn't, then someone seeing 11111110 might think it was eleven million, one hundred eleven thousand, one hundred ten instead of two hundred fifty four.

In the TI99/4A world, we don't use binary or octal so we don't need special

characters to alert us. Hexidecimal, however, is used extensively in assembly language for the TI99/4A and is represented by preceeding a numeral with a greater-than sign as in the following expression: >FE = 254.

Why use hexidecimal?

The first computers ever built used the binary number system extensively. This is because our computer circuits only recognize two values - call them on/off, up/down, high/low, 1/0, whatever.

Eventually, computers were built which could manipulate groups of bits all at the same time. Octal notation (base 8) became very popular because processors could handle eight bits at once. When TI (and others, later) moved to 16 bit microprocessors, hexidecimal notation got a big boost.

Please realize that at the computer's level, things are still 0's and 1's. The advantage to using octal or hexidecimal notation is that these numerals can be more easily converted into binary than can decimal. For example, given the following table of numerals:

BINARY	OCTAL	DECIMAL	HEXIDECIMAL
0000	0000	0000	0000
0001	0001	0001	0001
0010	0002	0002	0002
0011	0003	0003	0003
0100	0004	0004	0004
0101	0005	0005	0005
0110	0006	0006	0006
0111	0007	0007	0007
1000	0010	0008	0008
1001	0011	0009	0009
1010	0012	0010	000A
1011	0013	0011	000B
1100	0014	0012	000C
1101	0015	0013	000D
1110	0016	0014	000E
1111	0017	0015	000F
10000	0020	0016	0010

... how would you represent the number sixty five thousand, four hundred seventy nine in binary? In base ten, it is 65479. In octal it is 177707_8. In hexidecimal it is $FFC7_{16}$ or simply >FFC7. In binary it is 1111111111000111_2. (Trust me ....)

To convert from octal or hexidecimal to binary is easy! Looking at the chart above, we can convert each octal digit directly into a string of three binary digits:

$1_8\ \ 7_8\ \ 7_8\ \ 7_8\ \ 0_8\ \ 7_8$ =
$1_2\ 111_2\ 111_2\ 111_2\ 000_2\ 111_2$
or 1111111111000111_2

Similarly for hexidecimal:

$F_{16}\ \ F_{16}\ \ C_{16}\ \ 7_{16}$ =
$1111_2\ 1111_2\ 1100_2\ 0111_2$
or 1111111111000111_2

This still doesn't explain why we deal with hexidecimal. The reason is that when you get down to the bare bones of the machine (the registers) many things get done either at the bit level or in groups of bits. For example, if you want to change the video registers (change the screen color, go from 32 to 40 columns, or whatever) you will have to load one of the video registers with a certain bit pattern.

For example, video register 7 sets the background color of the screen (and the foreground, too, if you are in 40 column mode). The register is eight bits wide and 'looks' like this:

```
 BACKGROUND   FOREGROUND
+-------------------------+
|  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+
```

The first four bits (first starting at the left) are the background color and the last four bits are the foreground color. Let's say that you want the screen to have white text on a cyan background. The color codes are 8 for cyan and 15 (>F) for white. How do you shoe-horn these values into the register? You have to get the 8 value into the first four bits - to do that you have to multiply by 16. In other words, to load the register use 8x16 + 15. This isn't difficult - but it's not necessary.

If you used the hexidecimal version of the color codes - all you would have to do is "glue" them together to get your register value - >8F.

Remember that the TI99/4A is built around the hexidecimal number system. The microprocessor can operate on eight bit values and representing register values is a lot easier with two hexidecimal digits than it is with eight binary digits! As a final example, consider setting video register #1 which determines the following video characteristics:

Bit 0 - 4/16K memory (always set to 1 - all TI99/4A's have 16K of video memory)

Bit 1 - Screen display enable (1) or disable (0). When disabled, the screen is blank.

Bit 2 - Interrupt enable/disable. When set (1) the video chip will generate an interrupt every 60th (or 50th if you're running on 50Hz) second.

Bit 3 - Mode bit 1. When set (1) you're in text (40 column) mode.

Bit 4 - Mode bit 2. When set (1) you're in multi-color mode (seldom used).

Bit 5 - Reserved (for possible future use?)

Bit 6 - Sprite size selection. Set (1) means 16x16 bit sprites. Reset (0) means 8x8 bit sprites.

Bit 7 - Sprite magnification selection. Set (1) means double size. Reset (0) means normal.

Assume for a moment that you want to be in 32 column mode (i.e. not in 40 column mode, and not in multi-color mode). That means bits 3 and 4 should both be zero. You want double size sprites (16x16) but youdon't want them magnified. Therefore, bit 6 should be 1 and bit 7 should be 0. Finally, you DO want the user to see the screen (bit 1 set to 1), you want the video chip to generate an interrupt every 60th of a second so the sprites will move (bit 2 set to 1), and you know that bit 0 is always 1 and bit 5 is always 0. This gives a bit pattern of:

1 1 1 0 0 0 1 0

In base ten, this would be 226 - but I wouldn't want to calculate this each time I changed bits! In hexidecimal, it is:

1110 0010

E 2 or simply >E2.

(Continued from page 1)

The catalog will be updated as we receive new software. If you have some software that you would like to demonstrate and/or donate, please remember to bring it on a single sided, single density (SS/SD) disk. Please label the disk clearly and indicate what is required to run the program (Basic, Extended Basic, Editor Assembler option 3 or 5, etc.). For each disk of software donated to the club library, you will be reimbursed with a fresh blank disk.

In addition to all the software demos, the club witnessed a live demonstration on disk repairing by Curtis Provance. Vince Demers brought in a disk with a corrupted disk header and asked for some help to restore the disk and save the information that was still on the disk. After giving an impromptu overview on TI's disk structure, Curtis proceeded to demonstrate how to check and then repair the contents of the disk header using a sector editor program, such as John Birdwell's Disk information. The disk was successfully restored to its original condition right before our eyes. It seems this was a very popular topic based on the level of interest and all of the questions. We will try to repeat this demonstration at a future meeting.

In other news, Curtis gave us an update on his project to develop a cartridge dumping program. He has the main part of the code working but there are still one or two bugs in it. He showed several examples of cartridges he has successfully dumped to disk. Once saved to disk, these cartridge programs can be loaded and run from Extended Basic with a special loader program. Curtis is also working on this and will explain more about it at a future meeting.

All in all, it was a very fun and informative meeting. Now that I am all relaxed, I guess I should go back out and shovel some more snow. See you at the next meeting.

SCHEDULE OF MEETINGS

The next club meeting is scheduled for Monday February 19 starting at 6:30 PM. Meetings are held the third Monday of each month at the Science Enrichment Encounter (SEE) Center, 324 Commercial Street, Manchester, NH. Below is a list of dates for upcoming meetings. Annual dues are $15 payable to the New Hampshire 99'ers User Group.

February 19
March 19
April 16
May 21
June 18
July 16

NH99'ers User Group
PO Box 5991
Manchester, NH 03108
603-672-0084