# NEW HAMPSHIRE 99'ers

FEBRUARY 1988
NEWSLETTER VOL 6 NO. 2
POB 5991 MANCHESTER, NH 03108


**>OLD**

Last month's meeting was consumed by several software demonstrations. Let's start with Richard's:

The first disk that comes to mind is the Woodstock - Christmas animated cartoon. This program is an Extended BASIC marvel which does an **excellent** job with character redefinitions, sprites, colors - the works! This animated cartoon will really pick up your spirits. From the flapping of Woodstock's wings, to the sound of snow crunching underfoot, this program is detailed and quite entertaining. Richard ran out of copies at the last meeting; hopefully he'll have it again in March. Please **PLEASE PLEASE** (did that get your attention?) send a donation to the author (identified in the program) for the time put into this.

Another disk which Richard brought contained some of the most dazzling graphics I have ever seen on a TI. Anyone interested in developing graphics on the TI could learn some tricks from these people (a European firm selling games - I think). Hopefully, it too will be at the next meeting.

Disk #3 was a game - or maybe an adventure. It is called Freddy and comes to us from West Germany. Freddy is trapped in a maze of caves and must gather up as many treasures as he can before being poisoned from rat bites and other nasties. There are ropes to climb, ledges off of which you may jump (or fall), nasties, and treasures. The graphics are excellent and the tempo is challenging. Richard has converted it to English, so it is sehr leicht zu erlernen! (very easy to learn!)

Finally, Richard raved about the latest version of Disk Utilities from John Birdwell. It edits, catalogs, adds comments, copies, renames, slices, dices, chops - and if you order before midnight.... Whoops! Wrong commercial. Seriously, if you use DM-1000 and DISKO and ... throw them away. You will find this program as powerful and quite user friendly. This program may be purchased for $20 from:

John Birdwell
7252 Springhill Cir
Eden Prairie, MN 55344

My turn! My turn!

You all know Jim Peterson - the founder of Tigercub and a most generous man with his tips, shorts, one liners, and application programs. He has put together over the years three disks which he calls Nuts 'n' Bolts (#1, #2, and #3) which contain **hundreds** of subroutines in MERGE format. These subroutines do just about everything you would ever want - from redefining character definitions, to creating titles, clearing the screen in several different ways, to he offers. If you missed his demo disks at the last meeting, I'm sorry, but I will be sending them back. I ask you to trust me on this one: if you ever wanted to jazz up your Extended BASIC programs, this package is for you. Each disk is $15 and contains over one hundred subroutines - some of them are mini-programs! If you think that 15 cents a routine is too much to pay, then I suggest you calculate how much your time is worth when you are struggling to get that routine to work 'just the way' you want. There are also many other disks available including numerous games. A catalog is $1 with the $1 deducted from the first order. Please contact Jim now at:

Tigercub Software
156 Collingwood Avenue
Columbus, OH 43213

**>NEW**

The BCS Fayuh will be held the second weekend of April in the Lexington area (I assume). Our club plans to be there to sell some disks. We would like to bring our club system up to date with the purchase of two half-height, double sided (and double density) drives. Many club demos have been Murphied by disk incompatibility. Besides, our current SS/SD lone drive has seen better days. Please bring your most interesting disk(s) to the next meeting so that we can choose which disks we will offer. Obviously, all material has to be public domain.

We also have elections coming up in June. Start thinking about how you can help the club. There are numerous functions which in themselves are quite easy and consume little time. However, when you have to do a few dozen of them yourself, they begin to add up. We are fortunate to have several people helping us in unofficial capacities, as well as the four elected officers. Please make a commitment to give something to the club instead of always just taking. You'll be rewarded in the long run.

Articles?.... I was tempted to not put together a newsletter again this month. Come on, people! Am I the only person still interested in learning about the TI? Don't any of you play adventure games? Create graphics? Exercise utility programs? If you want me to edit your articles, I will do that happily. The only thing I ask is that I get the article on a disk (which will be returned). I may even put some stuff on the disk for you before I return it....

**Thanks for the memories....**

So long to Mike Mannion, a former vice President, and a good friend of the club. Mike donated much effort to put out the newsletter each month. Mike also added many programs to the library and was often the first one on the block with a new program or piece of hardware. Thanks again, Mike, for all you've done. We'll miss you!

## Infinite Sounds

Curtis Alan Provance
Paragon Computing

In Extended BASIC and BASIC, the CALL SOUND subroutine requires that you specify a time duration (a little over four seconds max) and you must access registers in sequence - i.e. to access register 3, you must also send values for registers 1 and 2. Both of these 'features' may be easily discarded by redefining CALL SOUND in a subroutine. To do this requires expanded memory and Extended BASIC. Note that the CALL LOAD's may also be performed in BASIC with a cartridge or DSR card which supports CALL LOAD's.

This redefinition provides access to each of the three sound registers independently and does not automatically turn off generation. In other words, once you have turned on a register, you must turn it off with another CALL LOAD. The system will also turn off all registers after a BEEP or HONK. This means that INPUT, ACCEPT ... BEEP ... and any error reporting will reset all registers (turn them off).

The CALL SOUND redefinition is fairly short:

```
30000 SUB SOUND(REG,FREQ,VOL)
30001 A=111860.8 :: B=INT(A/FREQ/16)
30002 CALL LOAD(-31744,96+32*REG+A/FREQ-16*B)
30003 CALL LOAD(-31744,B)
30004 CALL LOAD(-31744,128+16*REG+VOL/2)
30005 SUBEND

30006 SUB NOISE(TYPE,VOL)
30007 CALL LOAD(-31744,223+TYPE)
30008 CALL LOAD(-31744,240+VOL/2)
30009 SUBEND
```

### HOW IT WORKS

The sound chip is 'memory mapped' meaning that talking to the chip requires that you send data to a specific address, in this case -31744 (in hex code - >8400). It also means (at least on the TI) that you can only read or write one byte at a time - a byte representing any integer from 0 to 255. The sound register requires that certain bits be set, in this case, the bit that represents integer 128 (I'll explain the 96 in line 30002 momentarily). Other bits tell the chip which register is being accessed and what the following bytes mean - frequency or volume. In the case of the frequency, there are more than eight bits of information, so two bunches must be sent.

Here's how the bits tell the chip what's going on:

```
1000xxxx - (128) Frequency for register 1.
1001xxxx - (144) Volume for register 1.
1010xxxx - (160) Frequency for register 2.
1011xxxx - (176) Volume for register 2.
1100xxxx - (192) Frequency for register 3
1101xxxx - (208) Volume for register 3.
1110xxxx - (224) Noise control.
1111xxxx - (240) Noise volume.
```

Note that the register frequency control differs from one to the other by 32. The register volume control differs by 16. Therefore , to access register 1, we need to send 128 (=96 + 32*1) for frequency control and 144 (=128 + 16*1) for volume control. Of course, there are other bytes to be sent as well.

The frequency code consists of ten bits and is computed by dividing 111860.8 by the frequency. At this point, the **LAST** four bits must be sent with the register frequency access code; the highest six are sent in the next byte.

The volume byte is simply the access code plus the volume/2 (I'll explain later).

A full description of the program follows:

Line 30000: REG is the register number: 1, 2, or 3. FREG is the frequency with limits of 110 to 55,938. Volume ranges from loudest=0 to off=30. Note that the volume is divided by two. Although both BASIC's provide you with a volume range of 0-30, the sound generator actually can only recognize 16 different values (0 to 15). If you remember to use only 0 to 15, you can eliminate the '/2',

Line 30001: Since the value 111860.8 shows up four times (if you count appearances in B) I decided to assign it to A. By taking the integer of this value divided by the FREQ and 16, we are effectively eliminating the last four bits, i.e. isolating the highest six bits.

Line 30002: This is the LOAD that sets the access code for the frequency register (the 96+32*REG part) as well as sending the lowest four bits of the frequency code. A/FREQ gives all ten, while 16*B gives the same high six bits with the four lowest bits set to 0. Subtracting gives just the low four bits.

Line 30003: Yes, B is the value of the high six bits, divided by 16 (that's the way the chip wants it!)

Line 30004: The volume register code is 128+16*REG and the volume is VOL/2. Remember that you are free to eliminate the '/2' portion is you use just 0 to 15 as volume codes.

Line 30006: Type is the type of noise (1 to 8) as described in both BASIC's. Volume is the same as in sound.

Line 30007: Although the BASIC's make you use values of 1 to 8, the noise register only accepts values from 0 to 7. The normal noise type code is 224, but I subtracted one to fix your noise type.

Line 30008: Nothing new with the volume code here.

### WARNING!

While this technique works well with the sound chip, it will not work with the video chip or GROM chips. This is because the BASIC and Extended BASIC interpreters use both the video and GROM ports while executing your commands. You may have luck accessing the speech device - I frankly have never tried.

## STAND-ALONE ROUTINES

Curtis Alan Provance
Paragon Computing

Stand-alone routines allow your assembly program to run properly without the aid of CALL INIT or routines built into cartridges. Some programs - in an effort to save memory - still insist on pulling data or instructions from the ROM (or even GROM!) of a cartridge. This is a dangerous practice which must be avoided now that so many loading options exist for PROGRAM IMAGE files.

Many newsletters have published 'stand-alone' routines for your use, some specializing in speed, others concentrating on compactness of code. There is a happy medium that I strive for when I write my stand-alone routines. There are a few programming tricks which are widely accepted, while other tricks are 'boo-hoo'ed by 'serious programmers.' One of these tricks is self-modifying code.

Self-modifying code means that one portion of your program will actually change the machine instructions in another portion. This can be quite serious, particularly if there are 'bugs' anywhere - who knows what code you'll have when you're done? Used judiciously in manageable block sizes, this code can compact your program, free valuable registers, or allow instructions to use registers when they might otherwise require separate data locations. Let's take a look at a few examples:

### SUB-ROUTINES

Suppose you have a few subroutines which are called from various parts of your program. Furthermore, let us constrict them by saying that they are not recursive in any way, that is, they never branch and link to themselves, or branch and link to any other subroutine(s) which might branch and link back on the original. This isn't too bad of a restriction, and it fits most subroutines (at least the ones I write!) If you want recursion, you'll need a stack which I will address in another article.

Some of the subroutines may have to branch and link to other subroutines. How do you save the return address each time? You could move R11 to another register:

```
SUB1
        MOV  R11,R12     Save return address - 2 bytes
        BL   @SUB2       Branch and link to another
        B    *R12        Return - 2 bytes
```

This only uses four bytes of memory for program control and would be fine, except for two limitations which really hamper development:

1) You will run out of usable registers if you 'nest' your routines too deeply.

2) Each routine in a series must have its own dedicated register, so that no two related routines will attempt to store their return addresses in the same register.

We could use storage spaces for each routine, something like what the Editor/Assembler book recommends:

```
SUB1SV EQU  >8300       Address stored here - 2 bytes
SUB1
       MOV  R11,@SUB1SV Save address - 4 bytes
       BL   @SUB2       Branch and link to another
       MOV  @SUB1SV,R11 Restore address - 4 bytes
       RT               Return - 2 bytes
```

Wow! We just tripled the amount of memory needed to keep track of program flow - from four bytes to twelve (counting the two bytes at >8300). On top of that, we have to deal with the extra execution time of the MOV instructions. Well, we'll have to save R11 somewhere other than a register, so let's accept the MOV R11,@SUB1SV as a given. We can tighten up three other lines, tho':

```
SUB1
       MOV  R11,@SUB1SV Save address - 4 bytes
       BL   @SUB2       Branch and link to another
SUB1SV EQU  $+2         Points to @@@@@ - 0 bytes
       B    @@@@@       Return - 4 bytes
```

We are down to eight bytes of memory for program control, versus twelve with the previous method. We also have removed the execution time of a MOV instruction, tho' I admit that the new Branch will take a little longer than the register version. Saving four bytes hardly seems worth the effort, until you start adding up the number of routines you have - the TI isn't generous on memory, so use as little as you possible.

The first two examples are 'ROMable' code meaning you could burn them into a chip and they would function properly. Alas, example three is self-modifying and therefore not 'ROMable' - doomed to be forever executed from RAM. If this is acceptable to you, then I recommend example three over the other two.

I have another example of code which modifies itself depending on the routine to be performed. This is my latest attempt at a tight set of video routines. While it may not look any tighter on the surface, I can assure you that this saves quite a few bytes of memory over other versions. It also allows the easy implementation of a CLEAR, HCHAR, and SCROLL routine.

```
VSBR    DATA VDPWS,VSBR1        These video routines function identically to
VMBR    DATA VDPWS,VMBR1        those described in the Editor/Assembler manual.
VSBW    DATA VDPWS,VSBW1        "
VMBW    DATA VDPWS,VMBW1        "
VWTR    DATA VDPWS,VWTR1        "
CLEAR   DATA VDPWS,CLEAR1       Takes no arguments
HCHAR   DATA VDPWS,HCHAR1       R0 = address, R1(high byte) = byte, R2 = count
SCROLL  DATA VDPWS,SCROL1       Takes no arguments

* For most of the routines, R9 contains the VDP address; R10 contains the CPU
* address of the string (or the actual byte) and R12 contains the count

VSBR1
        LI   R12,1              We'll only read one byte
        MOV  R13,R10            Address of R0 of calling workspace
        INCT R10                Address of R1 of calling workspace
        JMP  VREAD              Generic read routine
VMBR1
        BL   @VMB               Load registers for multiple byte operation
VREAD
        MOV  R5,R0              Load 'DOIT' with MOV  R8,*R10+
        CLR  R9                 Don't fiddle with address bits
        JMP  VDPDO              Generic video byte move routine
VWTR1
        LI   R9,>8000           Set high bit to signify register writing
        CLR  R12                Don't write any bytes after address is set
        JMP  VWRITF             Go to generic write routine
VSBW1
        LI   R12,1              One byte to write
        MOV  R13,R10            Address of R0 of calling workspace
        INCT R10                Address of R1 of calling workspace
        JMP  VWRITE             Generic write routine
VMBW1
        BL   @VMB               Load registers for multiple byte operation
VWRITE
        MOV  R3,R9              Set write bit in address register
VWRITF
        MOV  R4,R0              Set 'DOIT' to MOV  *R10+,*R6
VDPDO
        A    *R13,R9            Add caller's R0 to whatever bits were set (if any)
        BL   @VLOAD             Load the address register
VLUP
        DEC  R12                Count off each byte (VWTR uses 0)
        JLT  VRTWP              Don't move any more bytes - RTWP

* The video workspace actually contains some instructions and permanent data

VDPWS
DOIT
        DATA 0                  R0 - either read or write instruction will go here
        JMP  VLUP               R1 - loop back for next byte
VRTWP
        RTWP                    R2 - return with workspace pointer
        DATA >4000              R3 - used to set and reset the write bit
        MOVB *R10+,*R6          R4 - write from CPU to VDP
        MOVB *R8,*R10+          R5 - write from VDP to CPU
        DATA >8C00              R6 - VDPWD
        DATA >8C02              R7 - VDPWA
```

```
        DATA >8800              R8 - VDPRD
        BSS  14                 R9 through R15
VMB
        MOV  @4(R13),R12        Get count of number of bytes to read
        MOV  @2(R13),R10        CPU address of data (or byte in HCHAR)
        RT

* I set the write bit here for those routines which can use it

VLOADW
        SOC  R3,R9              Set the write bit
VLOAD
        SWPB R9                 Low byte must be loaded first
        MOVB R9,*R7             Load low byte of address into VDP address register
        SWPB R9                 Restore to original form
        MOVB R9,*R7             Load high byte into address register
        RT

* CLEAR ********************** Fills screen with space character

CLEAR1
        BL   @GETWID            Puts width in R12, screen end in SCREND

* R9 should still be zero - left over from the MPY  @DEC24,R9 in GETWID

        LI   R10,>2000          Use the space character
        MOV  @SCREND,R12        Address of last character (screen size-1)
        INC  R12                Correct count to cover address >0000
        JMP  HCHAR2             Let HCHAR do the rest

* HCHAR ********************** R0 = address, R1 (high byte ) = byte, R2 = count

HCHAR1
        BL   @GETWID            Puts width in R12, screen end in SCREND
        MOV  *R13,R9            VDP address where HCHAR is to start
        BL   @VMB               Get count and byte to be written

* If the ending address is greater than the end of the screen, HCHAR will wrap
* around to zero and finish loading characters.

HCHAR2
        BL   @VLOADW            Load the address register with write bit set
        SZC  R3,R9              Restore to normal address
HCHAR3
        DEC  R12                Have we written all the bytes?
        JNC  HCHAR5             Yes, return
        C    R9,@SCREND         Past the end of the screen?
        JGT  HCHAR4             No, it's OK to write this byte
HCHARC
        MOVB R10,*R6            Write the byte into position
        INC  R9                 Next address in VDP
        JMP  HCHAR3             Loop back to check for proper addresses
HCHAR4
        INC  R12                Adjust R12 to cover the DEC after HCHAR3
        CLR  R9                 Start over at >0000
        JMP  HCHAR2             We have to reset the video address register
HCHAR5
        RTWP
```

```
* SCROLL ********************* Moves all lines up one row, writes line of spaces

SCROL1
        BL    @GETWID             Puts width in R12, screen end in SCREND
        MOV   R12,R9              We'll start reading at the second line
SCROL2
        BL    @VLOAD              Load the address register
        MOV   R5,@SCRLDO          We'll be moving from VDP to CPU
        BL    @SCRLUP             Read in one line's worth of characters
        S     R12,R9              Point to previous line in VDP
        BL    @VLOADW             Load the address register with write bit set
        MOV   R4,@SCRLDO          We'll be moving from CPU to VDP
        BL    @SCRLUP             Move one line's worth of characters
        SZC   R3,R9               Restore R9 to actual address
        A     R12,R9              Point to line just moved
        A     R12,R9              Point to next line to be moved
        C     R9,@SCREND          Are we past the end of the screen?
        JLT   SCROL2              No, move this line back one row

* At this point, the VDP address is pointing to the first column in the last
* row and the write bit is still active.  We can now clear the last line.

        LI    R10,>2000           We'll use space characters
        MOV   @HCHARC,@SCRLDO     We'll use the same command as HCHAR
        BL    @SCRLP1             Write one line's worth of spaces
        RTWP
SCRLUP
        LI    R10,SCROLB          CPU address of SCROLL buffer
SCRLP1
        MOV   R12,R0              Make a copy of the width for use as a counter
SCRLDO
        DATA  0                   Operation will be stored here before called
        DEC   R0                  Count of one line's worth of characters
        JGT   SCRLDO              Loop until done
        RT
SCROLB  BSS   40                  Has to be able to hold forty characters max
DEC24   DATA  24                  Used to calculate the snd of screen address

* GETWID ********************* Determines the width of the screen from >83D4
* For this to work, the screen must be ON, i.e. >83D4 = 111mxxxx where m is
* the mode bit (GRAPH = 0, TEXT = 1) and we don't care about the x's

GETWID
        MOVB  @>83D4,R9           Copy of register #1, 1111xxxx or 1110xxxx
        ANDI  R9,>5000            R9 now equals >5000 (TEXT) or >4000 (GRAPH) mode
        SRL   R9,9                Make the screen width a word - >0028 or >0020
        MOV   R9,R12              Store this for use by the calling routine
        MPY   @DEC24,R9           Multiply by 24 to get the number of bytes
        DEC   R10                 Address of last character in screen
        MOV   R10,@SCREND         Total number of bytes in the screen
        RT
SCREND  DATA  >1234               Will hold the address of the last screen character
```

## MEMBERSHIP RENEWALS

Just a reminder to everyone. If your dues are not paid up by the March club meeting, your name will be dropped from the club mailing list. We simply cannot afford to keep sending out 35 + ++ee newsletters every month. Dues are $15.oo a year, payable to the club: "New Hampshire 99'ers User Group". Please help to support the club.

## THE COMPUTER FAIRE

At the last club meeting we agreed to participate in the upcoming TI Faire (second weekend in April) sponsored by the Boston Computer Society. Our goal is to raise money to finance 2 new disk drives for the club system. To do this we plan to sell disks. Bring your favorite demos, games, utilities, graphics, etc. to the March meeting and we will review them. The best disks will be selected and copies made to sell at the faire. If we can come up with 8-1Ø demo disks and make 1Ø copies each, we could sell them for $4-5 each and make approximately $4ØØ-$5ØØ. Of course, we could make more demos and more copies. It all depends on you! We need club support to make this happen. We need several volunteers to make copies of the various demo disks. We also need people to help out at the faire selling disks and giving demos. I will prepare a catalog listing of all of the disks that can be handed out at the faire. Even if we run out of disks to sell at the faire we could still take orders. I hope to have two full systems running demos at the faire to catch people's attention.

When you bring your demos to the meeting, please include a description of what's on your disk(s) along with a listing or printout of what files are on each disk. I need this info to make the catalog. We could also make copies of our club library listing to give out so people could order disks from our club library. That's just an idea. Bring any other suggestions or comments you might have to the meeting.

One last thought about the faire. I always find something interesting and new at every faire I go to. You can usually find lots of good bargains and great demos. I encourage you to go even if you are not helping with the club. For those helping with the club booth, let's have fun doing it!

NEW HAMPSHIRE 99'ERS USER GROUP

PO BOX 5991

MANCHESTER, NH 03108-5991


# PLEASE SUPPORT BONANZA - THEY SUPPORT US!