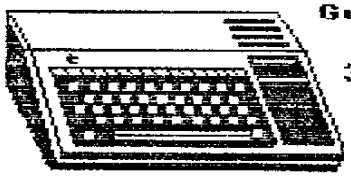
GUILFORD 99'ERS NEWSLETTER

SUPPORTING THE TEXAS INSTRUMENTS TI-99/4A COMPUTER



GUILFORD 99'ERS UG
3202 CANTERBURY DR
GREENSBORO NC
27408

TO:

VOLUME 8 NUMBER 3 MARCH 1991

George von Seth, Pres. (292-2035) Bob Carmany, Newsletter Ed (855-1538,
Tony Kleen, Sec/Treas (924-6344) Bill Woodruff,Pgm/Library (228-1892)
BBS: (919)621-2623 --ROS

The Guilford 99'er Users' Group Newsletter is free to dues paying members (One copy per family, please). Dues are $12.00 per family, per year. Send check to:Tony Kleen c/o 3202 Canterbury Dr., Greensboro, NC 27408. The Software Library is for dues paying members only. (Bob Carmany Ed)

OUR NEXT MEETING

DATE: March 5, 1991 Time: 7:30 PM. Place: Glenwood Recreation Center, 2010 S. Chapman Street.

Program for this meeting will be a program swap. We plan to have the UG library at the meeting and there will be other stuff available from private collections as well. Bring a supply of disks for the programs that you want!!

MINUTES

The meeting was brought to order (or at least a resemblence of order) by our President, George von Seth. There were 7 members present, and 2 guests. Introductions were shared with our guests and it was learned that they were attending our meeting in hopes that it was yet another club espousing the IBM PC. We apparently piqued their interest as they did stay, and participate in, the entire meeting.

First order of business, the minutes of last month's meeting, were read and approved. The second order of business was the treasurer's report. As of this writing, we have $148.71 in the treasury. Six members have paid their yearly dues.

New business topic: How do we recruit new members? Newsletter eidtor Bob Carmany proposed that we send postcards to all our old members informing them that we still exist, and that they are welcome back. I was drafted to draft the memo and send the invitations.

Another method would be to advertise via the community center's monthly promotional material. I'll review this possibility with the center's staff later this month.

Also suggested was MICROpendium support. We might place an ad. Or we might receive a mailing list. It was suggested that I might be able to check into that, too. I'm glad nothing else was suggested! (ha).

Bob Carmany presented our demonstration this month. His topic was "The Poor Man's Multiplan". He showed us how we could use TI-Writer (or the TIW clones) to sort, accumulate, cut, and otherwise edit a file similar to or at least with the same functionality as Multiplan. Several XB programs were used by several authors (Romer, Clulow, Wentzel).

As the title suggests, if you don't wish to shell out cash for Multiplan or a database software package; AND you have very little volumes of data; you can opt to use XB programs and TIW to manipulate this data for you! Thank you Bob for your fine demonstration (as Bob ALWAYS does).

Bob also whetted our appetite by demonstrating Funnelweb's 4.31 version. DISKREVIEW was Tony's answer to several file copying and sector editing programs. I plan to get my copy this month!

The meeting was adjourned, but the crowd lingered for nearly an hour afterwards sharing war stories (literally) and discussing TI99 possibilities and triumphs.

By the way, next month's program will be a disk swap. Bill, or whoever has the library by next meeting, will bring the club's diskettes. We'll have two systems at the meeting for copying your favorite software.

IF YOU HAVEN'T AS YET PAID YOUR DUES, YOU'RE DUE TO PAY THEM NOW!!!

Respectfully submitted,

Tony Kleen

TONY ON TIBASE

Use Topic - VDPram **/C Paging.
Tony Kleen, Guilford TI99er Users Grp

I've got two topics on VDPram paging. The first, presented here, is **/C paging. The '**/C' is an abbreviation for any command file. Next month's article will be on **/I paging; the '**/I' being an abbreviation for any install file.

I had planned to write about both kinds of paging (**/C and **/I) in this one article but decided it was too much to cover. I also suspect that Bob, our newsletter editor, would not care to devote an entire newsletter to just one article.

Okay, here's our problem. We've got the latest version of TIBase, V3.0, which allows us to execute our **/C files out of VDP memory. BUT... TIBase limits us to only 2546 bytes of VDPram for our use. AND... We've got more code in our */C files than will fit in memory at any one time. AND... We don't (as yet) own a RAMdisk.

Now that you've read the first few paragraphs, let me help you decide to read the rest of the article. Let's answer a couple of questions.

(?) Who would want to read the rest of this article? (*) Anyone who has a TIBase application (a collection of **/C files) that will not entirely fit in the VDPram memory. (*) Anyone who would like to improve the execute time of their **/C files to that of a RAMdisk.

(?) What do you expect to do after you read this article? (*) To execute all of your application's **/C files from the VDPram memory.

.....My basic examples.....

Immediately following are my three **/C files (BOOT, PROC1, and PROC2) that I'll use for my basic examples. You'll need to use your imagination to somehow assume that BOOT contains 500 bytes (characters) of information; that PROC1 contains 1000 bytes; and that PROC2 contains 1500 bytes. As the **/C files are listed here, they contain no where near those numbers.

```
* ---------------------------------- *
* BOOT/C (V1.00) basic example       *
* ---------------------------------- *
* (assume 500 bytes used)
*
LOCAL A C 1
WHILE 1=1
  CLEAR
  DISPLAY "Enter A character."
  READCHAR 22,1 A
  IF A="E"
    RETURN
  ELSE
    IF A="1"
      DO PROC1
    ELSE
      DO PROC2
    ENDIF
  ENDIF
ENDWHILE
* ----------------------------
```

```
* ---------------------------------- *
* PROC1/C (V1.00) basic example      *
* ---------------------------------- *
* (assume 1000 bytes used)
*
REPLACE A WITH " "
* ---------------------------------- *
```

```
* ---------------------------------- *
* PROC2/C (V1.00) basic example      *
* ---------------------------------- *
* (assume 1500 bytes used)
*
REPLACE A WITH " "
* ---------------------------------- *
```

.....VDPram.....

Version 3.0 has added an INSTALL capability which allows command files to be loaded into the VDP (visual display processor) and processed there as opposed to executing from a mechanical device. Executing a command file from VDPram is similar to executing from RAMdisk. Both execute from memory chips as opposed to mechanical devices. Executing from memory is, on average, twenty times faster than from diskette.

Multiple command files can be loaded into the VDPram area, as long as the 'condensed' files contain no more than 2546 bytes, the size of the VDPram. I say 'condensed' files because TIBase will strip the leading and trailing blanks (spaces) from every command line in your **/C file, prior to ADDing that file to VDPram. Also, comment lines are discarded.

The INSTALL capability also allows us to individually load and remove **/C files at will. If we wanted to, we could have a single **/C that ADDed and REMOVEd other **/C files at will! This **/C file could be thought of as the 'process' that will be doing our loading and unloading; the first requirement for 'paging'.

.....What is paging?....

Paging is the process of loading (and unloading) overlays into (and out of) memory. Let's break this definition down.

Paging is what? It is the "process of loading (and unloading)". We've just dicussed how this process can be done with the INSTALL capability of V3.0.

Load (and unload) what? We load (and unload) overlays. Again, INSTALL capabilities allow us to load and unload (ADD and REMOVE) command files. Overlays are synonymous with **/C files.

Into (and out of) what? The VDPram.

.....Paging **/C files.....

The INSTALL capability gives us every thing we need for paging. We create a single command file to process the loading and unloading of other command files into and out of the INSTALL's VDPram area. The single command file will load other command files with the INSTALL ADD directive, and will just as easily unload with the INSTALL REMOVE directive.

Back to our **/C examples. No modifications need to be done to any of the files except the one that is going to control the ADDing / REMOVEing; the BOOT.

```
* ---------------------------------- *
* BOOT/C (V1.01) control **/C paging.  *
* ---------------------------------- *
* (assume 500 bytes used)
*
LOCAL A C 1
WHILE 1=1
  CLEAR
  DISPLAY "Enter A character."
  READCHAR 22,1 A
  IF A="E"
    RETURN
  ELSE
    IF A="1"
      INSTALL ADD PROC1          <--
      DO PROC1
      INSTALL REMOVE PROC1       <--
    ELSE
      INSTALL ADD PROC2          <--
      DO PROC2
      INSTALL REMOVE PROC2       <--
    ENDIF
  ENDIF
ENDWHILE
* ---------------------------------- *
```

As you can see, we ADD the **/C file that is going to be executed by the DO directive, just before we execute it. Then, immediately after we've executed the **/C file, we REMOVE it from VDPram. By the way, the arrows are for display only. Don't put these in a command file.

We've still got one small problem. Our controlling **/C file, BOOT, ADDs and REMOVEs all our other **/C files, but BOOT/C remains on disk. How to get the BOOT itself into VDPram. Enter the following at the .DOT prompt:

```
INSTALL CLEAR
INSTALL ADD BOOT
DO BOOT
```

You have other alternatives, of course. You could add the first two directives to you SETUP/C file, and enter 'BOOT' or 'DO BOOT' at the .DOT prompt. Either way, let's assume you've started executing the BOOT command file.

At this moment, you've only got BOOT in VDPram. It uses only 500 bytes of VDPram, which gives us 2046 bytes of available space for loading other **/C files. Since PROC1 uses 1000 bytes and PROC2 uses 1500 bytes, for a total of 2500 bytes, we don't have enough space for all three **/C files in VDPram at the same time. We definitely want to execute our command files out of VDPram for the 20 fold increase in processing speed, so what do we do? We 'page' one **/C into VDPram when we need it, and 'page' it out when we're done.

When we page PROC1 into VDPram, we use 500 plus 1000 bytes, or 1500 bytes of VDPram. When we page PROC2, we use 2000 bytes of VDPram. Both values, 1500 and 2000, are below the VDPram space constraint of 2546!

.....So What?.....

What have we gained? (*) Well, for one thing, we now have a method whereby we can execute every one of our **/C files from VDPram. (*) Executing out of the VDPram gives us a 20-fold improvement over executing **/C files from diskette. If you do not own a RAMdisk, I highly recommand you use a paging technique.

There are some constraints to keep in mind. VDPram has only 2546 bytes of space. Any number of **/C files added to VDPram at the same time cannot exceed this limitation.

.....What if?.....

What if PROC1 needs 3000 bytes? Since BOOT uses 500 bytes, we only have room for a 2046 byte file. (*) Simply break the PROC1 into two or more smaller command files, each requiring less than 2046 bytes. BOOT would then need to be modified as follows:

```
* ------------------------------
* BOOT/C (V1.02) control several *
* ------------------------------
* (assume 500 bytes used)
*
LOCAL A C 1
WHILE 1=1
  CLEAR
  DISPLAY "Enter A character."
  READCHAR 22,1 A
  IF A="E"
    RETURN
  ELSE
    IF A="1"
      INSTALL ADD PROC1A         <--
      DO PROC1A                  <--
      INSTALL REMOVE PROC1A      <--
      INSTALL ADD PROC1B         <--
      DO PROC1B                  <--
      INSTALL REMOVE PROC1B      <--
    ELSE
      INSTALL ADD PROC2          <--
      DO PROC2
      INSTALL REMOVE PROC2       <--
    ENDIF
  ENDIF
ENDWHILE
* ---------------------------------- *
```

What if my application has 50 command files? Do I have to add/remove every one of those 50? (*) With **/C paging, YES!

The usefulness of **/C paging declines as you add more overlays to be paged. Your control **/C file requires 25 bytes every time you type the INSTALL ADD and the INSTALL REMOVE directives into it. If we had 50 **/C files, we'd need 50 times 25, or 1250 bytes, just to control our paging. That's half our VDPram area! What a waste!

But there is a better way. I currently have an application that uses 50 command files, yet all I have on my disk are 5 files that I page in and out. How do I do it? ... I'll show you next month.

TI-Base Topic - Menu Presentations.

by Tony Kleen, Guilford TI99er Users Grp

One of my requirements for a 'good' software system is that it be menu driven. You start with the master menu (the trunk of a tree?), which has its possible selections listed. You request one of the master menu selections (a tree's limb?) and are presented with more selections. You select one of these options (a tree branch?) and are presented another set of selections (another littlier branch?) or you may execute some small process (the tree's leaf?). You get the picture? of a tree? Menu driven systems can be likened to trees. You start with the trunk (the master menu) and keep branching out (the limb) until you finally get to the minute process (the leaf).

If you use the tree structure, you'll most likely be presenting a multitude of screens to the user. The purpose of this article is to demonstrate five alternative methods of presenting menu screens. I'll label these methods as follows: () the inline display; () the inline write; () the file list; () the database write; () and the database display.

First off, let's give a general idea of what the menu is going to look like:

```
:-----------------------------------:
:    MASTER MENU                    :
:                                   :
: 1 - Selection one                 :
:                                   :
: 2 - Selection two                 :
:                                   :
: 3 - Selection three               :
:                                   :
: H - Help                          :
:                                   :
: E - Exit                          :
:                                   :
:                                   :
:< >  Your selection, please.       :
:-----------------------------------:
```

What I'll be doing the rest of the article will be to present the command file code for each alternative, and follow that code with a brief discussion.

```
* ---------------------------------- *
* DEMO1/C  The Inline Display        *
* ---------------------------------- *
*
* (initialize)
LOCAL A C 1
*
WHILE 1=1
* (display the menu)
  CLEAR
  DISPLAY " MASTER MENU"
  DISPLAY " "
  DISPLAY " 1 - Selection one"
  DISPLAY " "
  DISPLAY " 2 - Selection two"
  DISPLAY " "
  DISPLAY " 3 - Selection three"
  DISPLAY " "
  DISPLAY " H - Help"
  DISPLAY " "
  DISPLAY " E - Exit"
  DISPLAY " "
  DISPLAY " "
  DISPLAY "< > Your selection, please "
  DISPLAY " "
  DISPLAY " "
  DISPLAY " "
  DISPLAY " "
  DISPLAY " "
  DISPLAY " "
  DISPLAY " "
*
* (User entry)
  READCHAR 14,2 A
*
* (process the selection)
  DOCASE
    CASE A="E"
      RETURN
    CASE A="H"
      DO HELP
    CASE A="1"
      DO PROC1
    CASE A="2"
      DO PROC2
    CASE A="3"
      DO PROC3
  ENDCASE
ENDWHILE
* ---------------------------------- *
```

There are four main segments to this command file: ()initialize; ()display; ()user entry; and ()process. I have defined one variable 'A' in the initialize segment.

You should notice that all but the 'initialize' segment lies between the (WHILE 1=1) / (ENDWHILE) directives. Using a 'WHILE 1=1' gives us an infinite loop. Not to worry, since this is what I want. We will continue to loop through these three segments (display, enter, process) until the user calls it quits, by selecting the <E>xit option. If the user selects one of the other options (1, 2, 3, or H), we process the appropriate DO directive. If the user selects an invalid option, we process nothing, and return to the DISPLAY segment of the WHILE/ENDWHILE loop.

Now to the DISPLAY segment. First off, I CLEAR the screen. This is optional, as everything on the screen is scrolled to the top, and off the screen; except for lines 23 and 24! If you don't CLEAR, and TI-Base has previously printed an error message, you won't have a clean screen.

As you can see from the coding, it takes twenty-one DISPLAY directives to complete the screen display. This amount of code is a major drawback if you're using the VDP memory, ie., the INSTALL ADD directive. You're using up ten to twenty percent of the VDP memory just to display one screen.

```
* ---------------------------------- *
* DEMO2/C  The Inline Write          *
* ---------------------------------- *
*
* (initialize)
LOCAL A C 1
*
WHILE 1=1
* (display the menu)
  CLEAR
  WRITE 1,2 "MASTER MENU"
  WRITE 3,2 "1 - Selection one"
  WRITE 5,2 "2 - Selection two"
  WRITE 7,2 "3 - Selection three"
  WRITE 9,2 "H - Help"
```

```
  WRITE 11,2 "E - Exit"
  WRITE 14,1 "< > Your selection,"
  WRITE 14,22 "please"
*
* (User entry)
  READCHAR 14,2 A
*
* (process the selection)
  DOCASE
    CASE A="E"
      RETURN
    CASE A="H"
      DO HELP
    CASE A="1"
      DO PROC1
    CASE A="2"
      DO PROC2
    CASE A="3"
      DO PROC3
  ENDCASE
ENDWHILE
* ---------------------------------- *
```

Again, we have the same four segments as the 'inline display'. Our advantage over the first alternative is in the 'display' segment and is that we write only those lines and columns that have information. We save VDP memory; again assuming we will want to use the INSTALL area. Also, as we are printing to the terminal's screen, it appears that we are painting the screen top to bottom. Some people like the 'painting' method better than the 'bottom to top scrolling' method.

Another advantage of 'painting' is that the user can be reading your screen text as the rest of the screen is still being printed.

```
* ---------------------------------- *
* DEMO3/C  The File List             *
* ---------------------------------- *
*
* (initialize)
LOCAL A C 1
SET PRINTER=DISPLAY
SET CRLF OFF
*
WHILE 1=1
* (display the menu)
  CLEAR
  LIST DSK1.DEMO3/L
*
* (User entry)
  READCHAR 14,2 A
*
* (process the selection)
  DOCASE
    CASE A="E"
      SET PRINTER=PIO.CR.LF
      SET CRLF ON
      RETURN
    CASE A="H"
      DO HELP
    CASE A="1"
      DO PROC1
    CASE A="2"
      DO PROC2
    CASE A="3"
      DO PROC3
  ENDCASE
ENDWHILE
* ---------------------------------- *
```

If you don't have version 3.0, the 'file list' alternative is not available to you. Notice that this alternative involves more initialization. We SET the PRINTER for DISPLAYing the printout at the terminal screen. The CRLF is also turned OFF; otherwise, we'd be double spacing the lines. Also, notice that the <E>xit process has changed. We have to reSET the PRINTER to our PIO.CR.LF, and reSET the CRLF to its ON status.

The two segments mentioned so far have minor changes compared to the display segment. One directive will display the entire screen!

Now for the downside. We LIST a file named DEMO3/L. This file has to be 21 lines long and 40 columns wide and must contain the information to be displayed, that is, that MASTER MENU screen at the bottom of the first column of this article. What we're doing is printing a file that contains the 840 screen display characters (21 rows times 40 columns). Also a downer, we need a different file for every menu screen we print. If we have ten menu's, we have 10 different list files. BUT, if you've only got one or two screens, this is a viable option.

```
* ---------------------------------
* DEMO4/C  The Database Write
* ---------------------------------
*
* (initialize)
LOCAL A C 1
LOCAL B C 1
USE DEMO04DB
SET SPACES=0
*
WHILE 1=1
* (display the menu)
  CLEAR
  REPLACE B WITH 1
  GO 0
  WHILE B<21
    WRITE B,1 LINES
    REPLACE B WITH B+1
    MOVE
  ENDWHILE
*
* (User entry)
  READCHAR 14,2 A
*
* (process the selection)
  DOCASE
    CASE A="E"
      CLOSE
      SET SPACES=1
      RETURN
    CASE A="H"
      DO HELP
    CASE A="1"
      DO PROC1
    CASE A="2"
      DO PROC2
    CASE A="3"
      DO PROC3
  ENDCASE
ENDWHILE
* ---------------------------------- *
```

Again, the initialization is different. This time, we have an extra variable, B. In addition, we're using a database named DEMO4DB. The database will contain one field, named LINES; which is 40 characters in length. We will require 21 rows per screen display, just like the 'list file' alternative. One other item in the initialization. We're SETting SPACES to zero, so that double spacing doesn't occur.

The display segment is different.

's an extra WHILE/ENDWHILE loop. need a counter, B, to tell us when we've printed 21 lines. Notice, too, that we GO to row number zero prior to displaying. To display the 2nd menu, we would GO to row number 21 instead.

I'll bet you're already getting ahead of me. A variation of this alternative would be to have two additional fields in the database; SCREEN-NUM and ROW-NUMBER. You could SORT the database ON SCREEN-NUMBER, and ROW-NUMBER. Assuming you're displaying SCREEN-NUM equal to 01, your logic would look like this:

```
FIND 01
WHILE SCREEN-NUM=01
  WRITE ROW-NUMBER,1 LINES
  MOVE
ENDWHILE
```

If you have two or more blank lines per screen, you'd be saving some disk file space on this variation. The process would run faster, too, since you're writing fewer lines.

****

```
* ---------------------------------- *
* DEMO5/C  The Database Display      *
* ---------------------------------- *
*
* (initialize)
LOCAL A C 1
USE DEMO04DB
SET SPACES=0
*
WHILE 1=1
* (display the menu)
  CLEAR
  GO 0
  DISPLAY 21 LINES
*
* (User entry)
  READCHAR 14,2 A
*
* (process the selection)
  DOCASE
    CASE A="E"
      CLOSE
      SET SPACES=1
      RETURN
    CASE A="H"
      DO HELP
    CASE A="1"
      DO PROC1
    CASE A="2"
      DO PROC2
    CASE A="3"
      DO PROC3
  ENDCASE
ENDWHILE
* ---------------------------------- *
```

****

Here, we've greatly simplified the 'display' segment. We just DISPLAY 21 LINES, after we position the database with the GO directive. Keeps it pretty simple, huh?

Now that I've presented all five alterntives, which do I recommend? Why all of them, of course. Each has its pluses and minuses. Each has it's own little niche. Actually, I would recommend the one you are most comfortable with, and understand how to use. If this helps you further in understanding and using Dennis Faherty's fine dbase product, I will have accomplished my purpose.

****

DISK DRIVES

By Jim Ness

It's funny (at least to me), but there are lots of people who seem to know lots of stuff about their computers, and all those tiny chips, and how the bits and bytes are handled. And there seems to be next to nobody that knows anything about disk drives, and how they work. Sensing this huge gap in man's knowledge, I decided to figure out what makes them tick.

The great thing about disk drives is that they can find files buried randomly within a huge field of data, and they do it pretty fast. Actually, they can do it so fast because it's not at all random.

The mechanical concept is not all that complicated. A small motor spins at 300 rpm (at least in this country, with its 60 hz power supply), and there is a tiny stepping motor attached to a read/write head. A stepping motor is a common item in indexing applications, where you want a motor to move a precise distance and stop on a dime. The read/write head is just a smaller version of what you have on a cassette recorder.

The stepping motor "steps" the head from track to track on a diskette. The tracks are concentric circles, not a long spiral as you would have on an album.

All of this is ultimately controlled by the disk software provided with your computer. Usually this is located in ROM within the machine. In most machines, the ROM is only sophisticated enough to load in the official Disk Operating System (DOS) which is located on the disk in the drive when the machine is turned on. The DOS contains all the file handling software, copying software, etc, and because it is on disk, it can be easily modified and/or updated as time goes by.

Our friends at TI decided to put the whole thing in ROM, which has a few bad side effects. First, it makes it hard to update and improve the software, which is located in the Disk Controller Card. Second, although the machine is a 64k machine, just like all the others, TI has set aside so much memory for special purposes, that there is only 32k left to play with. They set aside 8k for cartridges, 4k for disk drive, 4k for RS232/PIO cards, 4k for the Operating System (can't complain about that one), and 8k for various interfaces (speech, sound, VDP). Ok those are all good applications to have, but if you don't use them, you still can't use that memory for other things.

Anyway, all of the controlling software for the TI99/4A is located in the ROM card, as I said. This software tells the step motor when to step to the next track, when to return to the beginning, etc.

There is no standard for how a computer keeps track of data. In the case of TI, there is a directory of existing files, and a map of where they are located, at the beginning of each disk. These files are not necessarily all in complete groups. If you delete a 12 sector file from a disk, there is a 12 sector gap recorded in the map. Then if you add a 20 sector file, the software will put the first 12 sectors in the gap, and put the rest in the first available spot. When you ask for a file that is broken up this way, you can hear the disk head scooting along to read each individual segment.

Because the disk drives themselves are pretty standard, there are a few things that don't change. For instance, there are 48 tracks per inch in most 5 1/4" systems (There is a new 96 TPI system around, not TI compatible). And most systems only use 35 or 40 of the available 48 tracks. There are either 9 or 18 sectors per track (single or double density). Each sector holds 256 bytes of data. And the standard design allows 250,000 bits per second to be written.

Now, you say, 250k! That is about 25k bytes per second, right? How come I can not load a 25k pgm in one second, then?

Oh, yes, I mentioned that most drives are capable of transferring data at about 250,000 bits per second. And you were asking how come your programs don't transfer that fast.

Two reasons. First, as I said, the transfer of data is actually controlled by the ROM software in the TI99/4A. And to be as good as it is, it had to be a little bit slow. Not REAL slow (anyone ever use a C64 disk drive?), but not as fast as it could be. The second reason also has to do with software, but it is a universal problem associated with single density storage.

The major difference between single and double density storage is the way in which the data is coded. In order for the software to keep track of where the read head is located on a particular track, there are clock or synch bits laid down with

the data bits. In the old fashioned single density format, a synch bit was laid down ahead of each "0" bit, so there were never two "0" bits in a row. That kept the software from getting lost if there were a lot of "0" bits in series. Putting all those synch bits on the disk took up a tremendous amount of space that should be used for data.

So, some genius came up with a way of encoding the clock bits in with the data bits, so that no unnecessary space was lost. Voila, double density storage was born! And double density, as used with the Corcomp software, is said to increase transfer speed by at least 80%, mostly because the number of bits to transfer is cut way down.

So much for the exciting story of double density versus single density. How about double sided versus single sided? Well, obviously, it requires two read/write heads in the drive. Did you know that when reading a disk, the software reads, first, a track from side one, then the opposing track from side two, and continues back and forth? You didn't know that? There is a simple reason for doing it that way.

The disk head needs something to keep the disk stationary against it. In a single sided drive, there is a small arm holding the back side of the disk against the head. In a double sided drive, that arm would be in the way of the back side read/write head, so the solution was to use the two heads, directly across from one another, to hold the disk in place. In order to keep them across from one another, they alternate reading or writing as I said above. Very interesting, right? So if you wreck one side of a dbl sided disk, you can kiss the whole thing goodbye.

COMMENTS

By Bob Carmany

How many times have you gotten out a disk with a seldom-used program, stuffed it into the disk drive, and encountered Murphy's Law. The usual scenario is enter data at a prompt and get an error message or see the program unceremoniously crash. After much head-scratching (and an expletive or two) you try again! Same result! Puzzled by this aberration, you do what you should have done in the first place ---READ THE DOCS!

The most frequent failing in getting a program to work correctly is the ominous "operator malfunction". In short, the program is OK--you screwed up! The same goes for hardware projects. Most of the problems that I have had constructing gear for my computer have arisen because I either got in a hurry or figured that the task at hand was so obvious that I didn't need to do more than glance at the documentation. WRONG!!

While I'm thinking about it, is anyone interested in classes of any sort about the TI? Assembly Language, Forth, Multi-Plan, BASIC, or just about anything else that you can think of? I'm sure that if there was some interest shown, we could get something together! There are all sorts of possibilities. If you have a request, see George or me at the next meeting.

We are also soliciting IDEAS for what you would like to see in the newsletter in the upcoming months. What programs are you having problems mastering? Are there any hints or shortcuts that you would like to see? Or, what kind of tutorial programs would you like to see in the newsletter. You don't even have to write anything --we'll do that for you. Bring your suggestions to the next meeting.

```
100 ! DIRECT SOUND CONTROL
110 ! DEMO PROGRAM
120 ! BY Tim MacEachern
130 !
140 ! DARTMOUTH, NOVA SCOTIA
150 !
160 !
170 S=-31744 !ADDRESS OF SOU
ND CHIP >8400
180 V1=0 ! VOICE 1 FLAG
190 V2=32 ! VOICE 2 FLAG
200 V3=64 ! VOICE 3 FLAG
210 N=96 ! NOISE FLAG
220 C=128 ! COMMAND FLAG
230 F=0 ! FREQUENCY FLAG
240 A=16 ! ATTENUATION FLAG
250 WHITE=0 ! WHITE NOISE FL
AG
260 PERIODC=4 ! PERIODIC NOI
SE FLAG
270 CALL INIT :: CALL CLEAR

280 ! DEMO--START VOICE ONE
290 Q$="SET VOICE 1 IN THREE
 LOADS"
300 GOSUB 1010
310 CALL LOAD(S,C+V1+A+0)! S
ET ATTENUATION TO 0
320 CALL LOAD(S,C+V1+F+0)! S
ET BOTTOM FOUR BITS OF COUNT
DOWN RATE TO 0
330 CALL LOAD(S,33)! SET TOP
 6 BITS OF COUNTDOWN RATE
340 GOSUB 930
350 Q$="SET VOICE 1 IN A SIN
GLE LOAD"
360 GOSUB 1010
370 CALL LOAD(S,C+V1+A+6,0,C
+V1+F+0,22)
380 GOSUB 930
390 Q$="ATTENUATION DEMO"
400 GOSUB 1010
410 CALL LOAD(S,C+V1+A+6,0,C
+V1+F+0,0,56)! START VOICE O
NE AS A REFERENCE (VERY QUIE
T)
420 CALL LOAD(S,C+V2+A+15,0,
C+V2+F+0,0,48)! TURN V2 OFF
BUT PRESET TO ONE
430 FOR I=1 TO 5
440 FOR ATTEN=15 TO 0 STEP -
1
450 CALL LOAD(S,C+V2+A+ATTEN
)
460 FOR DELAY=1 TO 40 :: NEX
T DELAY
470 NEXT ATTEN
480 NEXT I
490 GOSUB 930
500 Q$="COUNTDOWN RATE DEMO"
510 GOSUB 1010
520 CALL LOAD(S,C+V1+A+1)! S
ET V1 ATTENUATION
530 FOR RATE=0 TO 1028 STEP
16
540 FOR BOTTOM4BITS=0 TO 15
550 CALL LOAD(S,C+V1+F+BOTTO
M4BITS,0,RATE/16)
560 NEXT BOTTOM4BITS
570 NEXT RATE
580 GOSUB 930
590 Q$="CALCULATION OF RATE
FOR     MIDDLE C(FREQUENCY 2
61.63)"
600 GOSUB 1010
610 FREQ=261.63
620 RATE=111860.8/FREQ
630 CALL LOAD(S,C+V1+A+0,0,C
+V1+F+(RATE AND 15),0,RATE/1
6)
640 GOSUB 930
650 Q$="NOISE CONTROL OPTION
S"
660 GOSUB 1010
670 Q$="WHITE NOISE TYPE 0"
680 GOSUB 1010
690 CALL LOAD(S,C+N+A+0,0,C+
N+F+WHITE+0)
700 GOSUB 930
710 Q$="WHITE NOISE TYPE 1"
720 GOSUB 1010
730 CALL LOAD(S,C+N+A+0,0,C+
N+F+WHITE+1)
740 GOSUB 930
750 Q$="WHITE NOISE TYPE 2"
760 GOSUB 1010
770 CALL LOAD(S,C+N+A+0,0,C+
N+F+WHITE+2)
780 GOSUB 930
790 Q$="WHITE NOISE TYPE 3"
800 GOSUB 1010
810 CALL LOAD(S,C+N+A+0,0,C+
N+F+WHITE+3)
820 FOR DELAY=1 TO 500 :: NE
XT DELAY
830 Q$="CONTROL NOISE TYPE 3
 THROUGH FREQUENCY OF VOICE"
840 GOSUB 1010
850 FOR I=1 TO 10
860 RATE=RND23
870 CALL LOAD(S,C+V4+A+15,0,
C+V3+F+(RATE AND 15),0,RATE/
16,0,C+N+F+WHITE+3)
880 FOR DELAY=1 TO 300 :: NE
XT DELAY
890 NEXT I
900 GOSUB 930
910 STOP

920 ! TURN OFF ALL VOICES
930 FOR I=1 TO 500 :: NEXT I
940 DISPLAY AT(12,1)ERASE AL
L:"TURN OFF ALL VOICES"
950 FOR DELAY=1 TO 300 :: NE
XT DELAY
960 CALL LOAD(S,C+V1+A+15)
970 CALL LOAD(S,C+V2+A+15)
980 CALL LOAD(S,C+V3+A+15)
990 CALL LOAD(S,C+N+A+15)
1000 RETURN
1010 DISPLAY AT(12,1)ERASE A
LL:Q$
1020 RETURN
```