TEXAS INSTRUMENTS

TI Forum

by Ron Albright and Jonathan Zittrain

Happy Anniversary...

October 28, 1983, when TI left the 99/4A marketplace, is now a vague memory to most. It was, early on, a scary event with disappointment and uncertainty. It is now an event remembered as a new beginning. Since then, the TI has been nurtured by many and pushed to heights even TI could not have imagined. It is a heritage all 99ers should revel in. Be proud. I, for one, salute you all. Happy Anniversary!

Just When You Think...

Just when you think the TI-related hardware/software is dwindling, new producers and new products appear. According to a live telecommunication conference held recently on Compuserve's TI Forum, Monty Schmidt (author of the popular Fairware "Techie" BBS system) and Ryte Data (210 Haliburton Street, Haliburton, Ontario, Canada K0M 1S0; (705) 457-2774) have teamed up to produce some exciting new software. As part of the transcript of the conference, here's how Monty himself described what is in the works:

(1) A new assembly-disassembly book. This book, untitled so far, is a complete disassembly of three peripheral cards the TI RS232 card, the TI Disk controller, and the Corcomp Triple-tech/9900 clock card. In the book, appx. 200-250 pages—will be complete comments, very detailed, and routines on how to access and use each of these cards at chip level, as of yet the routines are only for the assembly language programmer but hope to add some for the XB programmer also. (2) Super Clock Support—this is a program which accesses the Corcomp 9900 clock card and triple tech card. It adds a number of new calls from XB and accesses the card at a direct chip level. Features include three independently settable and readable elapsed timers. Reading the date in a format of Wednesday, January 1, 1986 rather than 3.01/01/86. Reading time in a 24 or 12 hour mode with A.M. P.M. indicator. Reading and writing time, date or day of week independently of each other, and lastly, two interrupt routines—INTA allows constant display of time on the screen, INTB allows calling time to the screen by use of the Control T key. It includes two files on the disk, one with all the features, and one with a chopped down feature for BBS users. The disk is completely unprotected and comes with a manual. (3) GPL-LINKER software which allows you to link and run output from the GPL-ASSM program, and run it in console without a grom-emulator. I've been writing programs in GPL and using the GPL-ASSM to assemble them and then use my linker to hook them to a GPL simulator which then runs them in memory expansion."

Obviously, some exciting products are in the works. A letter or card of encouragement to Ryte Data might be a good idea if you are interested in any of these new developments. Additionally, Rave 99 Company (23 Florence Road, Bloomfield, CT 06002: (203) 872-9272—after 6:00 pm EDT) has announced plans to produce an IBM-selectric-style keyboard for the TI. It would plug into the console keyboard connector but would allow the use of a full-sized 5151 keyboard layout with numeric keypad and lots more. The price is to be around $150. Write or call for additional details.

Asgard Software (P.O. Box 10306, Rockville, MD 20850; (301) 345-2492) continues to crank out some really useful software. "Pre-Scan-It" by J. Peter Hoddie (president, I think, of the Boston Computer Society's TI SIG) is a one-of-a-kind utility program. This beauty quickly rewrites Extended Basic Programs to execute faster and start almost instantly. It even "compresses" programs (in a sense) so that they take up less memory. It is well documented and a must for anyone who programs or uses XB programs. A real steal at $10. I hope to have a review of another new Asgard product, Graphx Companion III, next month. If someone will remind me.

Helo Awaits...

Have you ever wished you had someone to ask a question about the TI 99/4A but you live away from a users group and don't have a modem to connect with Compuserve's TI Forum? Well, there is a viable alternative and guaranteed to be there. Dr. Guy-Stefan Romano has, since March, 1981, answered calls from around the world from TI users needing help. He will answer your calls from 9-4 PDT (116 Carl Street, San Francisco, CA 94117; (415) 753-5581) with answers to programming problems, software evaluations and availability of public-domain progams for your specific needs. Amazing, eh? The cost? Well, it will cost you whatever the phone call costs through Ma Bell. That's it. No strings. No sales pitch. No nothing. This genuis (and I do not use that term loosely), is sincerely there to help. Folks, I do not exaggerate when I say that Dr. Romano is the most knowledgeable person I have ever met. Not just about TI but about anything. A real Renaissance man. Call him or write him. You'll never regret it.

Toure de Boards

The "old cruiser" has been at it again. Thanks to PC Pursuit (Information 1-800-368-4215, voice). I (RA) have the pleasure and honor of checking out some really terrific TI Bulletin Boards in some of Pursuits cities. Pursuit, as you may know, is a service offered by GTE/Telenet that allows subscribers to call BBS' in any of 16 large cities (Atlanta, Houston, Los Angeles, Chicago, Seattle, Boston, among others) via Telenet. You have to hook up with a BBS there, no voice calls. You can connect as long and as often as you like between 6:00 pm and 7:00 am local time and weekends for a flat monthly fee of $25. No charge per call or for total connect time. You can do some serious cruisn' with this deal. Also, with the new system, its a direct connect—no more callbacks, etc. You call Telenet, input your access code, the area code and phone number you want and away you go. While presently Pursuit seems overloaded (the lines are often busy), they have promised to increase lines soon. I heartily recommend HUG (Houston Users Group) TIBBS (713-475-8909; Bill Knecht, Sysop), Phoenix TIBBS (714-537-0741; Bill Rister, Sysop), and three terrific Boston BBS's (all area code 617: 321-8214; 331-4181; and 335-8475. I can never keep these straight. Two of these boards are run by BCS and one by UGN; in any case, all three are super). I love Pursuit, even with the busy signals, and I love TI BBS's for the magnani-

continued on page 224

The Inner Limits

by Glenn Davis

People often criticize the Forth language for being backwards and cryptic. In the hands of lousy programmers that is often true. The same can be said for any language when its users don't understand it. LOGO is often hailed for being an easy-to-use childrens' language. Forth is more like LOGO than any other high-level language available for the TI. If you like LOGO, give Forth a shot. What Forth adds is speed and full-machine support.

As if you couldn't tell, Forth is my favorite language. Unlike some languages, namely Pascal, Forth has well-defined I/O commands which complement each other. Forth's stack helps to keep the number of "names" to a minimum. In Pascal, each and every value must have a name associated with it, even if it is just the index to a loop or a counter. But we're not here to hear me rag on Pascal. We're talking Forth.

The article on Forth reading materials in the June 1986 *Computer Shopper* neglected one ideal source for Forth stuff. The Milwaukee Area 99/4 User Group runs the International Forth Information Center (4122 North Glenway, Wauwatosa, WI 53222). The Center has reprints of articles, much like the LA 99'ers User Group, except that you get to choose which titles you want; they are available for just $0.10 per page. In addition, much of the code in the articles is available on disk (SSSD) for $3 per disk ppd. These disks are "chuck full" of Forth goodies. Service is fast too. My order arrived in about a week. To find out which topics are current, send a SASE to them and you'll receive a list of the items they carry. You'll be glad that you did.

Null Bug Fix

Just one (we hope) more bug remained hidden in TI Forth until recently. Forth could not read a null record from disk, though the disk controller had no trouble at all. When trying to write a Forth source<disk file (and vice versa) conversion routine I discovered that "blank" lines would crash the system. Since source normally has a good share of blank lines this was unacceptable. It turned out that the problem was in how TI wrote the file RD and WRT words. The byte-count returned in the PAB was used as an argument for VMBx (VDP Multiple Byte Read or Write). Hackers will be able to see what's coming...The VMBR and VMBW routines cannot be passed as ZERO. Internally, these routines go into a loop that wipes out VDP RAM, the VDP registers...

My solution was to recode the words RD and WRT as found in Screen #71. This screen can be used as an exact replacement for the system disks'. Both RD and WRT check to see whether the count is zero. If so, they skip the code that reads and writes to VDP RAM. There is another little-known problem with null records, however...

When trying to post messages on local BBSs using the text created by my Forth source converter the next bug appeared. Some terminal programs refused to accept line-lengths less than one. When the ASCII upload got to the "null" line, it crashed. The problem apparently is that when those terminal programs tried to "fetch" the text, they ran into the same VMBR problem. So to make this short story shorter, the converter program leaves "blank" lines with just one space in them. (This is what TI-Writer does. Use a disk editor to look at a file written by TI-Writer that contains

continued on page 94

The Inner Limits
continued from page 93

blank lines. The sequence 01 20 (hex) should appear where a blank line is). This program is easier to use than the one published in the *Shopper* in 1985, and the text files take up less space since trailing blanks are not saved. (If you would like this or any other program listing from The Inner Limits, send me an initialized disk (up to DSDD) in a self-addressed mailer with return postage c/o *Computer Shopper*. Forth source will be in text file for compatibility with other files. The Forth source converter requires two disk drives).

Auto BSAVE

Beginners often ask how to set up one of the "auto-load" Forth disks. It used to be somewhat difficult. Each time you recompiled you had to search your memory (or your vocabulary) for the load options to use. Recompiling is necessary whenever you modify something (like the editor) and want the new changes on boot. So I wrote screen #90 to recompile everything for me, load some other compiler control words, BSAVE it, and report the number of bytes left in memory.

This is what happens on screen #90, which is intended to be put there by users with more than 90K of disk storage (i.e. double-sided, double-density, or more than one drive). Single-sided, single-density disk users will have to put it somewhere else since their drive will only access up to 89. Try placing it on the blank screens before the 40-column editor. #90 forgets the ASSEMBLER vocabulary, which should be at the bottom of the dictionary. Next, the options that aren't in parentheses are loaded including the 40-column editor. Comment that out if you want the 64-column editor.

The "BLK @ 1+ LOAD" sequence will load the screen following the one it's on. If you put it on screen #90 like I suggest, then screen #91 will be loaded. The next line does the BSAVE. If you want the object code somewhere besides screen #92, change that value. The final line prints the remaining free memory in bytes. To activate the process, just LOAD the screen that you placed the code on. Somewhere on screen #3 (the boot screen) you must place 92 BLOAD so the whole "overlay" is booted too.

Several of the words on #91 are used by code I'll discuss shortly. Words that you may not have seen before. ASCII is used to place the ASCII value of the following non-space character on the stack, as in ASCII A to put 65 (decimal) on the stack. You can avoid looking up many character values this way. CONTROL is similar, CONTROL G puts a 7 on the stack. \ (backslash) skips the rest of the current line, as if it was completely commented out by ( ). \S skips the rest of the screen. THRU is one of my favorite words for testing new code. For example, 20 25 THRU will load 20, then 21, 22,... until 25. This is similar to --> but allows you to choose which screens to load on each "modify-test-edit" cycle. Try it for a while. If it doesn't suit your fancy, go back to -->.

Notice that these words have complete and obvious names. They are not abbreviated to the point of obscurity. \ will be used often so its meaning will be as clear as !. CONTROL is spelled out, not abbreviated as CNTRL or CTRL. Using names like these will only hinder readability. Don't abbreviate to save memory. If RAM is being eaten up, try reorganizing the code and making it more efficient and compact. Decompose it. Read *Thinking Forth* by Leo Brodie for ideas on decomposition. As a last resort, WIDTH (not SCRN_WIDTH) can be shortened to 5 (or so) instead of 31. To avoid a specific "collision," where two words aren't unique to 5 characters, WIDTH can be widened and then reset. Also avoid abbreviating to save source space. This is the number one problem with most published Forth source. When the name is too long, try choosing a better name.

Finally! Inverse Text

The screens marked "1st" through "3rd" give TI Forth the capability of using inverse text like the Apple II series computers have. Adding this to the TI-99/4A is possible because the TI was built as a versatile graphics machine. Whereas the Apple is stuck with whatever characters it has in ROM, the TI's are in RAM and can be modified easily. The TI can even emulate the AppleIIc's "mousetext" if desired.

To get the "inverse" and "normal" text on screen quickly, a technique called vectored execution is used. In simplistic terms, this means that we use one set of words to tell another set of words how to function. The first screen contains a word that initializes VDP RAM with character patterns for both the normal characters and the inverse characters (which happen to be the ASCII value + 128). The system utility GPLINK loads the character patterns at the address provided. GPLINK is called four times to load upper and lower case in both the "normal" area and the "inverse" area. Forth calculates the addresses where the patterns will reside. Since >800 in VDP RAM is where the pattern table begins, it is used as the offset in the definitions. Notice that all of the calculations are in brackets. This forces Forth to calculate them at compile-time instead of run-time (i.e. it is faster for the execution of VIDEO). LITERAL compiles these values into VIDEO. VIDEO is executed at the end of the screen to actually set the VDP up. NOTE: if you BSAVE the VIDEO word, it must be executed again (usually put on screen #3 following the BLOAD) to set up the VDP. Otherwise using inverse will have predictable bad results. Use VIDEO whenever you change video modes because that re-maps VDP RAM. VIDEO is not designed for bitmap mode, so stick to text or graphics-I. This is especially important when using the 64-column editor.

The second screen is the Forth assembler source for the code word EMIT' on the third screen. This is for your reference only, and doesn't need to be typed in. Examine it carefully to understand how it works.

EMIT' ("emit-prime") is much like the kernal's EMIT, except that it forces the high bit on for all characters over 1F (hex). Since we loaded the inverse patterns that way, all printable (not control) characters are printed in the inverse font. The ORI R2,>0080 machine instruction, written here in Forth parlance as 2 80 ORI, forces this bit on.

INVERSE forces the vectored execution. Since EMIT and EMIT' are in machine code, their Code Field Addresses (CFAs) point to the Parameter Field Addresses (PFAs). Given that knowledge, we write INVERSE to modify the CFA of EMIT to point to the PFA of EMIT'. All further

continued on page 224

```
Screen #71:

  ( FILE I/O ROUTINES 12JUL82 LCT)
  BASE->R HEX
  : DOI/O CLR-STAT PAB-ADDR @ VSBW PAB-ADDR @ 9 + 8356 !
    @ 837C C! DSRLNK CHK-STAT ;
  : OPN  0 DOI/O ;
  : CLSE 1 DOI/O ;
  : RD   2 DOI/O CHAR-CNT@ DUP -DUP IF PAB-VBUF @ PAB-BUF @ ROT
         VMBR  ENDIF ;
  : WRT  DUP -DUP  IF PAB-BUF @ PAB-VBUF @ ROT VMBW ENDIF
         CHAR-CNT!  3 DOI/O ;
  : RSTR REC-NO 4 DOI/O ;
  : LD   REC-NO 5 DOI/O ;
  : SV   REC-NO 6 DOI/O ;
  : DLT  7 DOI/O ;
  : SCRTCH REC-NO 8 DOI/O ;
  : STAT 9 DOI/O PAB-ADDR @ 8 + VSBR ;   R->BASE
```

```
Screen #90:

  ( loads system utilities from scratch  27Mar86 GED )

  FORGET ASSEMBLER

  -CODE -SYNONYMS  -FILE
  ( -TEXT -GRAPH -GRAPH1 -GRAPH2 -SPLIT )
  -BSAVE
  -COPY -DUMP -PRINT (  -64SUPPORT ) -EDITOR

  BLK @ 1+ LOAD
```

```
  ' TASK 92 BSAVE .
  CR FREE ." Bytes free"

Screen #91:

  ( Useful words to add to the TI FORTH compiler  25May86 GED   )
  BASE->R DECIMAL
  : FREE   ( --- bytes-unused)  SP@ HERE - . ;
  : PAGE   0 0 GOTOXY CLS ;
  : ASCII     ( ASCII char | --- n )   BL WORD HERE 1+ C@
    STATE @  IF  [COMPILE] LITERAL ENDIF ; IMMEDIATE
  : CONTROL  ( CONTROL char | --- n )   BL WORD HERE 1+ C@  31 AND
    STATE @ IF [COMPILE] LITERAL ENDIF ; IMMEDIATE
  : NOT    ( f --- -f )    0= ;
  : \      ( skip rest of line )
    IN @   C/L /  1+  C/L *   IN ! ; IMMEDIATE
  : \S     ( skip rest of screen )
    B/BUF IN ! ; IMMEDIATE
  : THRU   ( scr1a scr1e --- )    1+ SWAP  DO  I LOAD  LOOP ;
  R->BASE
```

```
1st Screen:

  \ set up TI VDP to act like the apple              25May86 GED
  BASE->R HEX
  : VIDEO
  \ clear character patterns 0-255
     800 [ 100 8 * ]  LITERAL  0FF VFILL
  \ set "normal" patterns (32-127)
     [ 800 20 8 * + ] LITERAL  834A ! 18 GPLLNK
     [ 800 60 8 * + ] LITERAL  834A ! 4A GPLLNK
  \ set "inverse" patterns (160-255)
     [ 800 80 8 * + 20 8 * + ] LITERAL  834A ! 18 GPLLNK
     [ 800 80 8 * + 60 8 * + ] LITERAL  834A ! 4A GPLLNK
  \ set "inverse" colors (invert bits to invert color)
     [ 800 80 8 * + 80 8 * + 1+ ] LITERAL
     [ 800 80 8 * + 20 8 * +    ] LITERAL
     DO 0FF I VXOR LOOP ;
  R->BASE      VIDEO

2nd Screen:

  \ alternate EMIT to be vectored to by other words  28Mar86 GED
  \S
  BASE->R HEX
  CODE EMIT'
       *SP+ 2 MOV,
        2  7F ANDI,
       2 1F  CI,
         GT IF,    2 80 ORI,
         ENDIF,
        1  -4 LI,
       14 @(UP) 8 MOV,
       8 *7 BL,
       42 @(UP) INC,
       NEXT,
  R->BASE
```

```
3rd Screen:

  \ alternate EMIT and vectoring words               28Mar86 GED

  BASE->R HEX
  CODE EMIT'
       C0B9 , 242 , 7F , 282 ,
       1F , 1501 , 1002 , 262 ,
       80 , 201 , -4 , C2E8 ,
       14 , 69B , 5A8 , 42 , 45F ,
  : INVERSE   \ sets output chars to black on white
     ' EMIT' CFA  ' EMIT CFA ! ;
  : NORMAL    \ sets output chars to white on black
     ' EMIT  DUP CFA  ! ;

  R->BASE
```

```
4th Screen:

  \ cursor positioning routines                      28Mar86  GED
  BASE->R

  : AT   ( row column --- )  SWAP GOTOXY ;

  : HTAB   ( column --- )
    CURPOS @  SCRN_WIDTH @  /  SCRN_WIDTH @  * +  CURPOS ! ;

  : VTAB   ( row --- )
    CURPOS @  SCRN_WIDTH @  MOD  SWAP
    SCRN_WIDTH @  * +  CURPOS ! ;
```

```
  R->BASE

5th Screen:

  \ center a string on the display                   25May86 GED
  BASE->R DECIMAL
  : MARGIN   \ put cursor at beginning of current line
    ( column ) @ HTAB ;
  : CENTER   \ cnt ---   place cursor so text will be centered
    MARGIN  SCRN_WIDTH @  SWAP - 2 /  0 MAX  HTAB ;
  : [C"]     \ addr cnt ---
    DUP  CENTER  TYPE ;
  : (C")     \ print quoted string at run-time
    R COUNT  DUP 1+ #CELLS  R> + >R  [C"] ;
  : C"  \ C" cccc"
    ASCII "  STATE @ IF  COMPILE (C")  WORD HERE C@ 1+ #CELLS
                        ALLOT
                   ELSE  WORD  HERE  COUNT  [C"]
                   ENDIF ; IMMEDIATE
  R->BASE
```

TI Forum
continued from page 93

mous syspos they represent.

TI "Snapshots...

This will be a new feature if you support it. It will feature short "snapshots" of the famous and the not-so-famous in the TI community. Those who are doing unique things with, to, and for their TIs and the TI colony. So send in your vignettes to the TI Forum at the *Shopper*. This snapshot was submitted by a TI personality in his own right, Howie Rosenberg of Farmingdale, NY, and a gifted programmer and "doer." He presents a friend who has long been inspiration to him and an example of what can be done with the unique features of the TI 99/4A. Howie writes:

"Angelo Morris bought his first computer, a TI-99/4A in 1980. Like many of us he learned to program, first in BASIC then Extended BASIC, and subsequently in other languages. Today Angelo is quite a capable programmer in BASIC and Extended BASIC, and despite several problems is achieving the same status in other higher level languages such as Forth and C. Quite a common story so far but there is one fact that makes it a story. Angelo has been blind since shortly after birth.

When Texas Instruments pioneering efforts in speech synthesis were applied to the first home computer, new horizons were opened up to folks like Angelo. A friend volunteered to tape the Users guide and BASIC manual. Using the speech synthesizer Angelo's first task in additon to learning BASIC was to devise BASIC screen reader programs that could be incorporated in any program and which would read the screen. His early learning and programming efforts utilized this technique. WHEN TEXT TO SPEECH contained in the TEII module was released, Angelo's horizons were broadened. TEXT to SPEECH has the ability to list programs via the synthesizer as well as speak words rather than the spelling to which he had previously been limited. Unfortunately, punctuation is not handled by TEXT to SPEECH so that he still made use of the screen speller technique. The disk based version of TEXT to SPEECH afforded him easier access to Extended BASIC but the program always had a tendency to lock up and did not have the program list capability so that for the most part spell techniques were more useful to him.

One of the greatest boons to Angelo was the first TEXT to PROGRAM (programs to convert a Display/Variable 80 to D/V 163 mergeable file). In addition to all the things that it accomplished for the sighted programmer such as the ability to transfer programs via modem, the ability to edit text and subsequently convert to text allowed him to readily use global replace for changing print and display statements to PRINT #N; thus allowing him to add speech to other people's software.

As Angelo tells it, he is not into Arcade games and graphics. His primary uses of the machine was for managing a small business and for managing his personal affairs. He has written much of the software he uses such as checkbook management, record keeping software, and modified much other software tailoring it for his own use. He has been involved in telecommunications from the very beginning. Telecommunications is a natural medium of communication for the visually impaired. The data bases available afford him a wide variety of information difficult for him to obtain otherwise.

Recently he has expanded his capabilities by switching to a VOTRAX type of synthesizer. While the speech quality does not compare to that of the TI synthesizer, the device has features which are extremely useful for a blind person. Any file that is sent to the VOTRAX as a destination is read.

Punctuation reading can be switched on and off and the speed of the device can be controlled. It is surprising how an unsighted person can recognize speech at a speed where everything sounds like a quacking duck to others. He also makes use of a speaker connected to the modulator input (there is no monitor attached to his computer). The screen refresh information that the speaker provides enables him to keep track of such things as prompts and the status of things while telecommunicating.

He now also owns a Xerox CP/M machine for which he also is programming. As of late he is investigating the world of C, in this case first on the Xerox and recently on the TI which as it is for many of us, an old friend which has afforded us and particularly Angelo a window into a new world."

Thanks, Howie for starting off the "TI Snapshots." Any more submissions?

To Protect Or Not...

As long as there has been software, there has been protection that has evolved since the first 99/4 rolled off the assembly line. "SAVE CS1, PROTECTED," or "SAVE DSK1.filename, PROTECTED" was a built-in routine of the TI Extended BASIC module that anyone could use while storing a program. When loaded, the program saved with the ", PROTECTED" option would respond with "* PROTECTION VIOLATION" if any kind of LIST or SAVE command would be "poked" or changed. A simple CALL LOAD address was found that would undo the Extended BASIC protection, and already the program protection race was on. At that time, user group newsletter editors were faced with a dilemma of whether or not to publish the CALL

continued on page 226

The Inner Limits
continued from page 94

calls to EMIT actually execute EMIT'. This is necessary because all kernal output routines call EMIT, such as . (dot) and TYPE.

If NORMAL didn't return things to normal, then all text printed after executing INVERSE would be black on white. NORMAL simply restores the CFA of EMIT to point to its own PFA. Elegant. Leo Brodie's DOER-MAKE construct is somewhat similar.

A few other effects are worth mentioning. INVERSE may cause some printers to display italics when using SWCH. This may be a benefit if you've wondered how to get Forth to do it! Also, Forth's EMIT will never print inverse text (ASCII A 128 + EMIT will have the same effect as ASCII A EMIT) because it strips the high bit off everything it emits (FIG-standard). To emit arguments this way without using INVERSE and NORMAL, just use EMIT8, which outputs all eight bits.

Another short set of powerful words are on the 4th and 5th screens. AT, HTAB, and VTAB are all cursor movement words. GOTOXY is not always convenient. Their use should be clear so I won't discuss them.

Automatic Centering

These words are very powerful. With them, you may specify any string and have it centered on the current screen line. For example, C" This is centered" puts it in the middle of the row the cursor is on. Operation is quite simple. MARGIN puts the cursor at the left edge of the screen (it must be modified if you intend to use it for the printer). CENTER moves the cursor to the correct position for centering the text based on the count. [C"] does the actual work via TYPE. (C") is a run-time-only routine (which is why the parentheses are there). It calculates the address and length of the string and passes them to [C"]. C" either compiles the string or prints it depending on whether it is being compiled or interpreted. C" is especially useful for printing program titles, prompts for data, menu titles and similar things. [C"] and C" are fully compatible with the inverse words.

Notice that all of these definitions are no more than two lines long, except for VIDEO and C", which could have been if the comments, calculations, and whitespace were removed. This is how Forth is supposed to look. Writing words like these in BASIC or Pascal would be difficult and syntax would be more complicated. Also notice that each word has a stack comment and often an effect comment. You should get in the habit of writing comments at the same time you write the code itself. Without a valid stack comment no programmer can "fix" your code.

Word sets like the one that contains VTAB and HTAB demonstrate the usefulness of THRU. There will be times when the centering routines are not wanted and the cursor movement routines are. THRU allows only the 4th screen to be loaded (and not the 5th). In this case, a single screen is a small deal. The tune changes when many word sets are spread over the disk. Connecting the screens with --> would force the centering words to be compiled whenever the cursor words are. This is a good example of the "modularity" of Forth and why "screens" are used in the first place. As a programmer, you can pick and choose what already-written code you wish to use instead of re-inventing the wheel repeatedly. Until next time...THE INNER LIMITS. ●

TI Forum
continued from page 224

LOAD address. The same was true with the TI Disk Manager's "proprietary protection" capability, where a disk could be protected from duplication with the Disk Manager module by anyone who entered <FCTN>X ten times at a Disk Manager prompt. A simple change with any disk sector modifier program could remove this protection, and as assembly language "sector copiers" evolved (where each sector of a disk could be copied, one by one, without regard to its contents) it became unnecessary to even bother removing the protection.

TI placed most of its software into "solid state software," or "command modules," which were made copyable only after TI pulled out of the 99/4A market. Yet other third party software producers, using disks or cassettes as the software media, were faced with a problem.

Unless a special protection were placed on the software, anyone could easily copy it. A person could copy a program and give it to a friend—who would no longer need to buy the program for himself. And that friend knew two other friends, and gave them copies as well.

The latest protection schemes have been broken in large part by "track" copiers, the next generation of disk copy programs. Many track copies have been made public domain or fairware by their authors, a move at once praised and condemned by different sectors of the TI community.

Software producers used different methods to deal with software piracy, Chris Bobbitt, manager of Asgard Software in Rockville, Maryland, said that his company has always supported a policy of marketing its programs unprotected. "I just don't think it's worth the time to devise new protection schemes that will only be broken a month later," Bobbitt said. "I never figure anything is lost to piracy. Pirates don't buy programs, anyway. They simply hoard them."

Craig Miller, owner of Millers Graphics in San Dimas, California, takes a different view. "Track copiers put one hellacious dent in our income," Miller said. "When they [the track copiers] hit [the general public], sales of Advanced Diagnostics and Explorer took a 50% nosedive. Now, there's no way to validate that [track copiers as the cause], but I think there's a relationship."

According to Miller, most Millers Graphics programs have been given the best protection scheme possible at the time. "The protection won't last long, but it prevents the casual copying of software, and keeps sales going longer, I believe," he said.

TI consumers had mixed reactions to manufacturers protecting their software. A legitimate user who buys a piece of protected software cannot make a backup copy, which he is by law entitled to do.

Howie Rosenberg, a 99/4A user from Long Island, New York, commented on CompuServe's TI Forum, "An author protecting his work and saying "take it or leave it" is no different than TI keeping so many secrets from us for so long...no one has ever said that TI did not have the right to sell their wares on an as is basis. Can we set up different rules for the [third party] software author?"

Miller agreed that his protected programs could not be moved to a RAM disk or other peripheral stage device, but could offer no direct solution. "It's an unfortunate situation," he said. As far as the original disk going bad on the user, leaving him without a backup, Miller said that he supports a free replacement policy. "Send us the disk, and we'll send out another one, within 48 hours."

Ironically, Millers Graphics' "Gram Kracker" made it possible for TI users to backup and copy their command modules. "I talked to both TI and Triton [a company selling the remainder of TI's stock of modules]," Miller said. Considering that for the price of a Gram Kracker ($180) one could buy every module in the Triton catalog, Triton had no problem with the production of the Gram Kracker, he said. If TI were producing new modules, he would probably not have released the Gram Kracker, he added.

The central question for software producers appears to have been, "Is program protection worth the trouble it causes the users, considering that the protection can rapidly be broken by the pirates?" "Definitely not," said Bobbitt. "Give the users reasonably priced software and they won't pirate it."

Paul Gray, another TI Forum member, agreed. "Some of these folks make it [track copiers] sound like the end of the world. The best way for them to stay in business though is to sell a QUALITY product at a REASONABLE price and provide AMPLE documentation. Most don't."

Many of the software producers for other computers have been attempting just that. Ashton-Tate, for example, no longer attempts to protect its software from copying. Rosenberg contends that the 99/4A world must be considered differently. "There are practically no new users or software producers joining us. In [other computer] worlds, a new user gets his machine and buys some software...before he "gets wise" and starts stealing it."

Such a closed system could mean that most active TI-99/4A users are now out of the habit of buying their software, and may instead pirate it. Also, software manufacturers for other computers almost uniformly charge higher prices for their software, and can absorb any losses due to piracy. 99/4A programs have been mostly below $40.

Bobbitt says the quality-for-a-low-price attitude is all it takes. "I decided I saw enough garbage getting sold for $39.95, and vowed I could do it better. My company is set up to destroy any LEGITIMATE reasons for piracy. Asgard started out as a statement."

If the questions for software producers was whether or not to protect their software, the question for consumers is whether or not to buy protected software.

"We are not dealing in a situation in which there is a definitive answer," said Rosenberg. "I feel that I for one was willing to give up my "rights" to freely copy software in order to keep some of the software producers."

Is such a sacrifice by the user necessary? Is it the honest user's job to pay for the problems caused by pirates?

A combination of product support good documentation, program updates to registered users, good quality, and low prices eliminate the need for program protection, say some.

"I have to eat," said Miller. "I've spent four years devoted exclusively to the TI-99/4A." If 99/4A software were not his only livelihood, he added, "it would be a whole different story [in regards to program protection]."

I asked both Bobbitt and Miller if they planned to continue their current protection policies on future programs. "Definitely," said Bobbitt, "although based on the wishes of the author, we will continue to encourage non-protection and complete customer support."

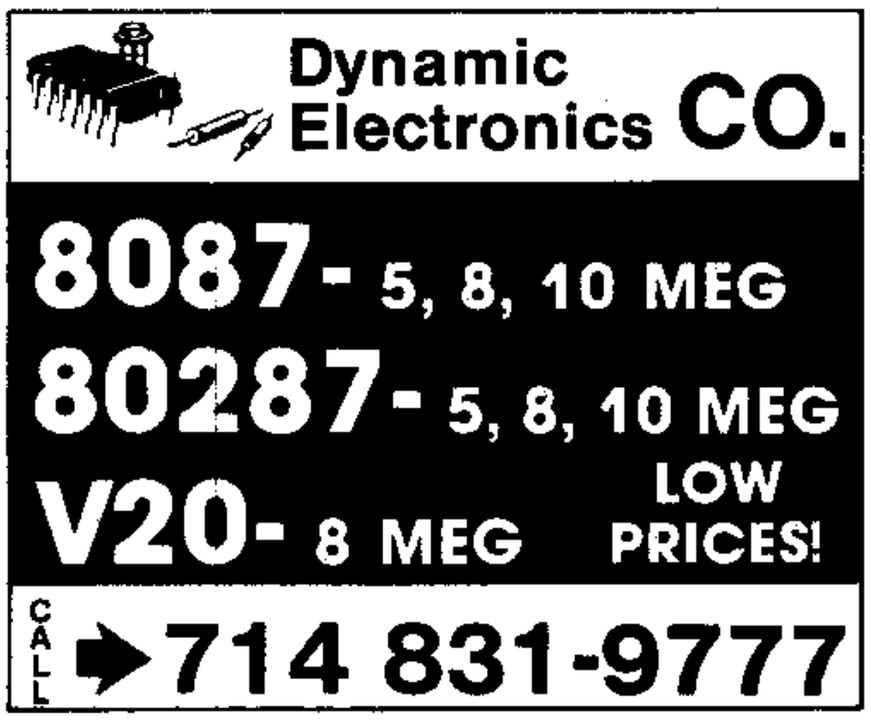
"Currently, yes. But I don't know if they're going to BE any future programs," Miller said. "There may come a day when there's nothing left to copy."

Forth Bits From Howie Rosenberg...

Howie also wants to share, periodically, some short Forth tips. He promises to continue if there is interest. But to show interest, you'll have to write us at the *Shopper*. Here is Howie's first installment.

Forth Bits is not a tutorial on Forth. There are a number of those around, several of which are quite good. The best "tutorial" on Forth is Leo Brodie's *Starting Forth* used in conjunction with the TI Forth manual which enables the user to establish the differences between TI and the version used in Brodie's book (and they are many). Forth Bits will be a collection of short tips, suggestions and code which is designed to aid those who have already started in Forth at least to the extent of reading a text such as Brodie's book and the TI manual but for one reason or another are hung up somewhere and cannot make progress in writing code in Forth. For some the hangup appears to be in the fact that Forth, in addition to the normal considerations of syntax and constructs, which are the two things a programmer usually need concern himself

continued on page 227



TI Forum
...ontinued from page 226

...ith when tackling a new ...nguage, requires that the ...ogrammer also make use of ...id keep track of) the sack ...d do all arithmetic opera...ons in Reverse Polish Nota...ion (RPN). As the intent of ...orth Bits is to encourage pro...grammers who have not to ...date used Forth to any great ...xtent to add it to their arsenal ...f languages, for it is a power...ul language with unique prop...rties that make it extremely ...seful in a variety of applica...ions. The Bits are not ar...ranged in any local sequence. Together in the sequence presented they do not constitute a text or a set of tutorials but a collection of hints which ...opefully will be of help in ...earning and using Forth.

...orth Bits 1—Writing and Debugging Forth Programs

As you are already aware, having read both *Starting Forth* and the *TI Forth* manual), there are two ways of entering Forth words. Code can be entered from the keyboard either as a set of direct commands, e.g.

4 5 + .

...ields an immediate result 9, or by typing in a new word for compilation, for example:

: ADD4/5 4 5 . ;

which compiles the new word ADD4/5 to the dictionary and makes it available until power down or execution of the FORGET word. The second, more permanent and generally necessary for sizable applications is, of course, entering the new words which constitute a program onto a Forth screen using the editor for incorporation into your Forth system using the LOAD word. I like to fool around at the console using the command mode. This is ideal for checking out new ideas and just getting ones creative juices flowing. For any serious application of size it is necessary, as in any language, to take a "top down" approach and look at the overall application and subdivide it from the top. While this form of programming is necessary in any language, in Forth it presents a seemingly philosophical contradiction. Forth is built up by adding words to the existing vocabulary. Whenever a new word is added, it can call all previous defined words including the kernel words and words the programmer has previously defined. All words in a new word definition must have been previously defined or an error results. Thus,

NEWWORD OLD1 3 OLD2 ;

simple executes OLD1 places 3 on top of the stack and then executes OLD2. Necessary is the previous definition of OLD1 a...d OLD2. The rule in generating Forth code is to conceptualize from the top down but in general code from the bottom up. Many times while generating and debugging Forth code it is necessary to test a concept or part of one from the top prior to coding a lower level word. This can be done by generating a do nothing word temporarily which can later be forgotten and replaced by a correct version of the word. Thus in the last example suppose OLD2 had not yet been defined. A "stub" is generated for debugging. : OLD@ ; NEWWORD is now defined. Obviously it will not do what it is supposed to do in the final application but many times the feasibility of sections of code can be tested in this way. Probably the most important single suggestion I can make to the Forth novice programmer to help him in debugging Forth code is simply when you debug a Forth word you should not only determine that the word does that which you have specified it do but CHECK THE STACK CONTENTS. Make sure that only those numbers which your design says should be on the stack indeed are on the stack. If you use the return stack for temporary storage determine that it has been restored after execution of your new word. Forth is actually one of the easiest languages to debug if you follow these simple rules. ●