THE NATIONAL NINETY-NINER

VOL II - NO. 9 - OCTOBER, 1985

$1.50

THE 99ER'S ASSOCIATION
3535 SO.H ST., #26
BAKERSFIELD, CALIF. 93304
(805) 397-4361
DON VEITH - EDITOR/PRESIDENT

CREATED FOR TI 99/4A HOME COMPUTER OWNERS

TABLE OF CONTENTS

THE NATIONAL NINETY-NINER

VOL II - NO. 9 - OCTOBER, 1985

ANNOUNCEMENTS

SUPERBUG II

By Edgar Dohmann

SUPERBUG II is now in distribution. I have substantially revised SUPER-BUGGER which TI released to public domain and have released this new program to FREEWARE distribution. SUPERBUG II comes on a full SS/SD disk which contains 3 versions of SUPERBUG II, an Extended Basic Loader, and a 40-page manual in TI-Writer format ready for printing on your printer using the TI-Writer Formatter.

Since the program is distributed through FREEWARE, everyone who has a copy of the program is authorized and encouraged to make as many copies as they wish and give them to anyone who wants one. I have spent considerable time creating this program and writing the manual so anyone who wishes to purchase SUPER BUG II should send $8.00 (price includes a diskette, disk mailer, and postage prepaid) to the following address:

SUPER BUG II
C/O THE 99'ERS ASSOCIATION
3535 S. H ST., # 93
BAKERSFIELD, CA 93304

If you do not know anyone who has a copy of the program, you may order one from me at the address above. I specifically request that none of the files on the SUPERBUG II distribution disk be put on any BBS system (including CompuServe and Source) for downloading. I want everyone who receives SUPERBUG II through FREEWARE to receive all files (including the documentation files) so the only authorized means of distributing the program is by disk copies. The distribution disk contains the following files:

SBUG	-- A Program File that loads at >A000
SBUG6	-- A Program File that loads at >6000
SBUGO	-- A Compressed Format Object File (Relocatable)
LOADSBUG	-- A Standard Format Loader for X-Basic
SBDOC	-- File 1 of the Manual (auto prints the other files)
SBDOCA	-- File 2 of the Manual
SBDOCB	-- File 3 of the Manual
SBDOCC	-- File 4 of the Manual

I will welcome the comments and suggestions of anyone who uses this program. My objective has been to keep the file size under 8K bytes in length. There are currently about 600 bytes left in this objective. It is possible that a future release may add additional features using this "spare" space. In the meantime, have fun with SUPERBUG II!

SUPER SPACE

By Edgar Dohmann

If you have been reading the June through August issues of the MICROpendium, you may have seen the series of articles on the SUPER CARTridge. Basically this is a how-to series on building your own cartridge with Editor/Assembler and 8K of CMOS RAM in the cartridge memory space. The only problem with SUPER CARTridge is that it requires modification of one or more existing cartridges to create the new cartridge. For many of us tinkerers this is no problem but I am sure there are a lot of people who would like the features of SUPER CARTridge without having to butcher up some existing cartridges.

As a result, I have made an agreement with Bill Moseid of Model Masters in California to produce SUPER SPACE. SUPER SPACE is essentially the same thing as SUPER CARTridge but is built on a specially designed cartridge board so no modifications are necessary. SUPER SPACE will have the Editor/Assembler GROM, 8K of CMOS RAM, and an optional battery. The target prices for SUPER SPACE at this time are:

KIT (without battery) ---- $29.95
KIT (with battery) ------- $32.95
ASSEMBLED (with battery) - $39.95

The kits will include all parts, the cartridge board, and a cartridge case along with assembly instructions. Only users who are capable of assembling kits that require soldering delicate integrated circuits should order SUPER SPACE in kit form. Model Masters warranty will apply only to SUPER SPACE products sold as assembled units.

If you are wondering what you can do with SUPER SPACE (or SUPER CARTridge for that matter), there are currently 2 disks of programs available for these products. One is my own SUPERBUG II which is in FREEWARE for $3.00 to $5.00 and includes a version that will load into the >6000 memory space. The other is a disk of software from David R. Romer, 213 Earl St., Walbridge OH 43465 which is available for $6.00. The July issue of MICROpendium also had instructions on how to set up a program with a GROM header for menu selection from a SUPER CARTridge (it will work the same with SUPER SPACE). I am sure that additional software will become available as more SUPER SPACE and SUPER CARTridge units get in the hands of users.

SUPER SPACE has already been prototyped and is working. The artwork for the cartridge board is in the process of being turned into circuit boards at this time so Model Masters hopes to have production units of SUPER SPACE available in October. Final price and ordering information will be available by the end of September.

FROM THE MAILBOX

THREE NEW CORCOMP PRODUCTS

CorComp, in a letter to Users Groups dated July 19, 1985, announced three new products. The three (3) new products include a 32K Memory Expansion, a Clock-Calendar unit, and Triple Tech. The 32K and Clock/Calendar unit are train components that plug directly into the 99/4A. Triple Tech is a card for the Expansion System that contains three functions.

Triple Tech contains a Clock/Calendar with a battery backup to allow operation of the unit for up to six months if the power is not turned on. The card also contains a 64K printer buffer compatible with any printer on the market. No user modification of software is required. The third element of Triple Tech is a Speech Synthesizer Connection. Simply remove the Speech Synthesizer Board from its housing and install it into Triple Tech.

The stand alone Clock/Calendar unit is housed in a 5" x 2" x 2 1/2" housing similar to the 9900 Micro Expansion unit. It includes battery backup to maintain accuracy of the clock and date functions. A Load Interrupt Switch, built into the unit, allows use of the newest screen dump programs available for the 99/4A.

The 32K Memory Expansion unit allows you to run Assembly Language, Logo, Pilot, and Forth programs. The unit provides only Expansion Memory with no ability for expansion to include other functions at a later date.

Suggested Retail Price for the products are: 32K Memory Expansion - $119.00; Clock/Calendar - $ 89.95; Triple Tech - $139.95. Contact CorComp directly or your favorite dealer for additional information on these new products.

ODDS 'N ENDS

THE TEN COMMANDMENTS OF HUMAN RELATIONS

1. Speak to people. There is nothing so nice as a cheerful word of greeting.

2. Smile at people. It takes 72 muscles to frown, only 14 to smile.

3. Call people by name. The sweetest music to anyone's ears is the sound of his own name.

4. Be friendly and helpful. If you want friends, you must be one.

5. Be cordial. Speak and act as if everything you do is a joy to you.

6. Be genuinely interested in people. You can like almost everybody if you try.

7. Be generous with praise and cautious with criticism.

8. Be considerate with the feelings of others. There are three sides to a controversy: yours, the other fellows, and the right side.

9. Be eager to lend a helping hand. Often it is appreciated more than you know. What counts most in life is what we do for others.

10. Add to this a good sense of humor, a huge dose of patience, and a dash of humility. This combination will open many doors and the rewards will be enormous.

The list above was copied from the Ann Landers column of May 25, 1985, issue of The BAKERSFIELD CALIFORNIAN.

FUTURE ISSUES

By Ben Veith, Editor

This is a new column which will appear on occasion. Its purpose will be to inform you of what we plan to have in the newsletter during the coming months. The January, 1985, issue will feature articles reprinted from Users Group newsletters. The three specific subjects covered are (1) TI Writer, Multiplan, and fan modifications for the Expansion System.

Another issue, in the 2nd Quarter next year, will feature reviews of all the available freeware software. Various staff members will be assigned to review software based upon their area of expertise. The list of available programs is so long and growing that this project may take more than one issue to complete. The regular articles by our staff of writers will continue to appear in each issue.

If you have any requests for an article on a specific subject, please write to me at our address on the Index Page. We are also seeking articles on any subject relating to the TI-99/4A. If you have an article of interest, please submit it on disk in a DIS-VAR 80 format. TI-Writer is used to prepare the newsletter for our printer.

ARTICLES

PASCAL NOTES

By Edgar Dohmann - JSC Users Group (JUG)

This month's article features a program I have been looking for or planning to write for some time now. It was written by James G. Schroeder of the Milwaukee Area 99/4 Users Group and was published in the July 1985 issue of his group's newsletter, HOCUS. The program is called DV80TOPAS and can be used to convert DIS/VAR 80 files to Pascal TEXT files.

```
PROGRAM DV80TOPAS;

LABEL 90,100,110,120;

CONST ENDSEC = 255;

VAR BLOCKNUMBER,COUNT,COUNT1,SECTOR,K,LEN,STRNGLEN : INTEGER;
    PAUSE : CHAR;
    FILENAME : STRING;
    F : INTERACTIVE;
    BUFFER1 : PACKED ARRAY[0..511 OF CHAR;
    BUFFER2 : PACKED ARRAY[0..79 OF CHAR;
BEGIN
    PAGE(OUTPUT);
    WRITELN('CONVERT DV80 TO PASCAL TEXT');
    WRITELN('ENTER FILE NAME FOR OUTPUT');
    READLN(FILENAME);
    REWRITE(F,FILENAME);
    WRITELN('ENTER STARTING SECTOR NUMBER');
    READLN(BLOCKNUMBER);
    BLOCKNUMBER := BLOCKNUMBER DIV 2;
    WRITELN('ENTER NUMBER OF SECTORS IN FILE');
    READLN(COUNT);
    COUNT := COUNT - 1;
    COUNT1 := 1;
 90 : STRNGLEN := 0;
      SECTOR := 0;
      UNITREAD(5,BUFFER1,511,BLOCKNUMBER,0);
      BLOCKNUMBER := BLOCKNUMBER + 1;
100 : LEN := ORD(BUFFER1[STRNGLEN);
      IF LEN = ENDSEC THEN GOTO 110;
      MOVERIGHT(BUFFER1[STRNGLEN+1,BUFFER2[0,LEN)
      WRITELN(F,BUFFER2:LEN);
      STRNGLEN := LEN + STRNGLEN + 1;
    GOTO 100
110 : IF COUNT1 = COUNT THEN GOTO 120;
      COUNT1 := COUNT1 + 1;
      IF SECTOR = 1 THEN GOTO 90;
      SECTOR := 1;
      STRNGLEN := 256;
    GOTO 100;
120 : CLOSE(F,LOCK);
END.
```

There are a few special rules that must be observed in order for this program to work properly. First, the D/V 80 file must start on an even numbered sector and must not be fractured. I suggest that you always use a freshly initialized disk to copy the D/V 80 file onto before running this program. If the D/V 80 file is the only one on the disk, you can be sure that it is not fractured and that it will start on sector 22 thus satisfying the requirements of this program without having to use some kind of disk fixer/analyzer program to make sure the source disk is set up properly.

The D/V 80 file must not have any control characters in it. If the text was entered using the Editor/Assembler's Editor then everything will be OK. If the text was entered from TI Writer however, the Save File command appends tab control characters to the end of the file. There are two ways around the problem: (1) Use Print File to save the TI Writer text instead of Save File or (2) Load the Saved TI Writer file back in with the Editor/Assembler Editor to get the tab control characters removed then Save the file again from Editor/Assembler.

The DV80TOPAS program should be on a disk in VOL. #4 and the D/V 80 text file must be on a disk in VOL. #5. The best thing to do is to insert a Pascal disk of some sort into VOL. #5 when you boot your system. This will allow the P-system to confirm that VOL. #5 is on line. Then when you are ready to run the program, remove the Pascal disk from VOL. #5 and replace it with the disk containing the D/V 80 text file. Make sure that there is adequate space on the disk in VOL. #4 that contains the DV80TOPAS program to also store the converted TEXT file when the program executes.

The program will first prompt you for an output filename. This should be a file on VOL. #4 and must be specified to be a TEXT file. An example of a proper response to this prompt is as follows:

```
#4:FILE.TEXT
```

The next prompt is for the starting sector number. This should be given in decimal and should be the starting sector of the file, not the directory sector for the file. If the procedure described above is followed in setting up the source disk, the proper response to this prompt will always be 22.

The final prompt is for the number of sectors in the file. You should use the Disk Manager to catalog the disk containing the D/V 80 file and remember the number of sectors that it indicates are used by the file. The number of sectors indicated by the Disk Manager will be the proper response to this prompt. As you probably know, the Disk Manager accounts for the file's directory header and lists 1 more sector on the catalog than is actually used by the file. This program expects you to enter the number which includes the directory and adjusts for it by decrementing the value of COUNT which you enter. Thanks again to James Schroeder. If anyone else out there has a handy utility Pascal program they would like to share with the rest of us, please drop me a line in care of this newsletter.

The TI Forth Dimension

Jeff Stanford

In my previous article, I discussed the different types of numbers used by Forth and how to write equations in postfix notation (or RPN). I also covered, briefly, the subject of fixed point numbers. I wish to discuss the example used in my last article before proceeding on to the subject of this article. In my example, I wanted to express five dollars and thirty five cents in a way to make it a whole integer and thus enter it into the computer as 535 (with a scaling factor of 100).

The point I failed to make was, instead of handling the number as a dollar amount requiring a factional part, it is more efficient in memory usage and computation speed to handle the number as a penny amount. In this manner, the amount may be expressed as a whole number (integer) and remove the need for floating point numbers. Of course, fixed point numbers may not right be for every application. The magnitude of the scaled number must be less than 32767 (or 2147483647 if you plan to use double precision integers). Therefore, if you plan work with a very large number, such as the 'National Debt', working with floating point numbers has its advantage.

Now on to the subject of this month's article: variables. TI Forth variable names have a lot of flexibility. They can consist from one (1) to thirty-one (31) letters in length and can consist of any character from the keyboard with the exception of the 'space' character (CHR$(32)) which is used as a delimiter. There are no reserved or protected names in Forth. If a previously defined variable is used, you will get a warning stating the name is not unique and access to the older version of the variable will be lost until the most recently defined word is deleted(1). Using variables in Forth is very different from Basic in two main ways. First (1) , before you can reference a variable, it must be defined. This is done by a word called 'VARIABLE' and is used in the following manner: ' initial_value VARIABLE name '. For example, we want to define two variables, X and Y, with initial values of zero. We would write:

```
0 VARIABLE X     0 VARIABLE Y
```

The foregoing statement reserves two two-byte memory locations with unique names that point to each other. I stress the word 'point' because that is the second (2) difference between using a variable in Forth and Basic. After your variable(s) are defined, they may be used for the storage of numbers. When a variable name is called in Forth, an address is displayed showing the variable's memory storage location. TI Basic returns the value stored for the variable rather than the storage address like Forth. Forth, unlike Basic, must utilize another word (command) to read or write values for a variable to or from its assigned memory address. Assembly uses a similar technique. A label assumes the address in memory where the number value of a vairable is stored. The Forth words used to read and write to memory are @ ('fetch') and ! ('store') and the way they are used is :

```
X @     to read from variable X
7 X !   to write 7 to variable X
```

Forth manuals and books define these two words by their effects on the stack. 'Fetch' requires an address to be on top of the stack which it replaces with the value found in that address (address --- value_in_address). 'Store' requires two numbers to be on the stack: the value to be stored and the address to which to store the number (number address ---). In each case you need to specify the address, which is left on the top of the stack when you use the variable's name, hence you will almost always find 'fetch' and 'store' used after a variable name unless the address is computed another way. As I said in my last article, Forth uses different types of numbers. So it follows that Forth also uses different ways to reference memory for each type. 'Fetch' and 'store' works on two bytes of memory at a time (16 bits). The other Forth words work the same way as 'fetch' and 'store' and only differ in the size of memory they work with. Here is a quick reference to show the different ways of accessing memory.

WORD	MEMORY SIZE REFERENCED	NOTES
C! (b addr ---)	1 byte stored	b is an eight bit number
C@ (addr --- b)	1 byte read	
! (n addr ---)	1 word stored (2 bytes)	n is a sixteen bit number
@ (addr --- n)	1 word read	
2! (d addr ---)	2 words stored(2)	d is a thirtytwo bit number
2@ (addr --- d)	2 words read(2)	
F! (fl addr ---)	floating point store(3)	fl is a sixtyfour bit number
F@ (addr --- fl)	floating point read(3)	

Another important word to remember when you working with variables is 'ALLOT'. As I mentioned above, e 'VARIABLE' only reserves two bytes for storage. More memory is required when you wish to work with double precision,floating point numbers, or more important, when an array is to be defined. 'ALLOT', when called by the computer, expects the number on top of the stack to be the number of bytes it must reserve for a variable. For example, a double precision number needs four bytes (32 bits). If 'VARIABLE' only reserves two bytes, we will still need two additional bytes for a double precision number. A definition for a double precision number would be like:

```
0 VARIABLE DBL-NUM 2 ALLOT
```

It goes likewise for floating point numbers which need eight bytes (64 bits) and a definition for a floating point number would be like:

```
0 VARIABLE FP-NUM  6 ALLOT
```

Arrays require a little more work because TI Forth doesn't have any predefined words to handle them. To define an array you count up how many bytes of memory you will need and then define your array like this:

```
0 VARIABLE ARRAY N ALLOT
```

N is equal to the number of bytes you need less two bytes. VARIABLE alloted the first two bytes that were required. NOTE: to prevent any problems you should always keep the number of bytes you reserve with ALLOT an even number. This is due to the the way 4A addresses memory. To access the array, we must compute the address of the element we desire. This is done by using the name of the array, in this case ARRAY, and adding the proper offset to get the address. For example, let's assume that our array consists of ten sixteen bit numbers. The address of the first element would be 'ARRAY 0 +' (ARRAY + 0), the second would be 'ARRAY 2 +', the third would be 'ARRAY 4 +', and so on. We are able to identify a pattern building in which the address of the Nth element would be 'ARRAY N 1 - 2 * +' (ARRAY + (N - 1) * 2), where the 'N 1 -' is simply used to convert from an array base of one to an array base of zero. The multiplication factor of 2 is due to our assuption assumed that our elements are sixteen bits in length or two bytes. If we were using byte sized elements, this number would be 1. If we we working with and array of floating point numbers the scaling factor would be eight (8). Therefore, if we wanted the address of the ninth element of 'ARRAY', it would be , using the equation above, 'ARRAY 9 1 - 2 * +' which becomes 'ARRAY 16 +' when we solve for the constants. If you desire to learn more about arrays, 'Starting Forth' by Leo Brodie has a good example of an application of arrays which is simple and easy to follow.

Another type of named memory storage used by Forth is the constant. It is, in simple terms, a variable which doesn't change its value. Forth does not treat constants the same way as it does variables. Instead of getting an address where the value is stored in memory on the stack, you get the value of the constant itself. Constants are defined in a manner similar to variables. The defining word is 'CONSTANT' instead of 'VARIABLE'. An example of its use is:

```
10 CONSTANT TEN
```

With this definition, everytime you use 'TEN' you will get the number 10 on top of the stack. For those who want more information and examples on using constants, any book on Forth will have all the information any one could desire.

For a closing note, every month I have people approach me trying to get copies of TI Forth or its manual. I assume there a lot of people still seeking to obtain both items. DO NOT GIVE UP!! MICROpendium is offering, in their freeware section, a complete TI Forth package including the system disk, manual, a disk with sample programs, and a disk with the assembly language source code. To take advantage of this offer, you simple send a check for twenty dollars ($20.00) and two initialized single-sided disks to:

```
      FORTH
c/o MICROpendium
    P. O. Box 1343
    Round Rock, TX 78680
```

MICROpendium supplies the third disk for the assembly source code. Next time I will try to tie everything together with some examples of graphics. Until then keep going Forth (sorry about that).

Footnotes

1 : For example, type this in ' : and hit enter. Forth will issue a warning saying '.' is not unique and then say 'ok'. Now every time you use '.' the screen will be cleared, a message will be printed, and then the number on the top of the stack will be printed. For example try ' 3 2 * yourself. Please make note that even though '.' is used in the definition of our '.', this is not a recursive call (a subroutine that calls itself). The only thing that happens is out new '.' is calling the older version of '.'. To delete the word just created and restore access to the older version of '.' type ' FORGET . enter '

2 : 2! and 2@ are not defined in TI Forth but are common enough to be listed. They can be added with the following definitions (TI FORTH, appendix C).

```
: 2! >R R ! R> 2+ ! ;
: 2@ >R R 2+ @ R> @ ;
```

3 : F! and F@ are added when the -FLOAT option is loaded from the menu

STRINGING AND UNSTRINGING

By Jim Peterson

The following program will give the plural form for most words. I will leave it to you to improve on it, and to teach the computer the correct plural form of PANTS, TOOTH, MOUSE, FUNGUS, DATA, and the other inconsistencies of the English language.

```
100 INPUT W$
110 L=LEN(W$)
120 Z$=SEG$(W$,L,1)
130 Y$=SEG$(W$,L-1,2)
140 ON POS("EFHNSXYZ",Z$,1)+
1 GOTO 320,150,180,210,240,2
70,270,290,270
150 IF SEG$(W$,L-2,2)<>"IF"
THEN 320
160 PL$=SEG$(W$,1,L-2)&"VES"
170 GOTO 330
180 IF (Y$<>"AF")&(Y$<>"LF")
&(Y$<>"RF")&(W$<>"HOOF")THEN
 320
190 PL$=SEG$(W$,1,L-1)&"VES"
200 GOTO 330
210 IF (Y$<>"CH")&(Y$<>"SH")
THEN 320
220 PL$=W$&"ES"
230 GOTO 330
240 IF SEG$(W$,L-2,3)<>"MAN"
 THEN 320
250 PL$=SEG$(W$,1,L-3)&"MEN"
260 GOTO 330
270 PL$=W$&"ES"
280 GOTO 330
290 IF (Y$="AY")+(Y$="EY")+(
Y$="OY")+(Y$="UY")THEN 320
300 PL$=SEG$(W$,1,L-1)&"IES"
310 GOTO 330
320 PL$=W$&"S"
330 PRINT PL$
```

This program is also a good example of four of the Basic statements which are used to manipulate strings. Remember that a string is a character, a group of characters, a word, a sentence, even a punctuation mark or a blank space or one or more numeric digits, which does not represent a numeric quantity. And a string variable name must end in a dollar sign.

Line 100 asks you to input a string. Line 110 contains the first of our string manipulators, LEN. All it does is to count the number of characters in the string, including blank spaces. Try it - type PRINT LEN("THIS IS A TEST") and Enter. So, L equals the number of characters in the string you input. Line 120 introduces the next string manipulator, SEG$. That stands for SEGMENT, and what it does is to pick a segment out of the string, starting at the position you specify and continuing for as many characters as you specify. So, SEG$(W$,L,1) selects the portion of W$ beginning at L and continuing for 1 character. Since we have just defined L as being the number of characters in the string, we are starting with the position of the last character and it would do no good to specify more than one character. So, Z$ is the last character of W$. Try it - type PRINT SEG$("TEST",LEN("TEST"),1).

Similarly, line 130 defines Y$ as being the segment of W$ starting with the character in the position of the length of the string, minus 1, and continuing for 2 characters - in other words, the last 2 characters of the string. Try that too - PRINT SEG$("TEST",LEN("TEST")-1,2). Line 140 introduces the manipulator POS, which means POSITION. It tells the computer to find the first occurrence, in the first string, of the character or characters in the second string, starting the search at the position specified. Try it - type PRINT POS("TEST","T",1). The answer is 1 because you told the computer to start searching for T, starting at the first character of "TEST", and "T" is the first character of "TEST". Try PRINT POS("TEST","T",2). Now the answer is 4, because you asked for the search to start at the second character, so the first "T" it found was the 4th character of "TEST". Finally, try PRINT POS("TEST","X",1). The answer is 0 because no "X" was found in "TEST".

Line 140 is a bit difficult to understand, but it shows one of the best uses of POS. It asks for the first occurence, starting with the first character of the string "EFHNSXYZ", of the string Z$ - which is the last character of the input string, remember? So, if W$ is a word ending in "E", Z$="E", and the POS of "E" in "EFHNSXYZ" is 1. If W$ ends in "Z" and Z$="Z", the POS of Z$ in "EFHNSXYZ" is 8 - get that? And if W$ ends in "W", Z$="W", and the position found by POS is 0 because it didn't find a W - remember trying that in the last paragraph.

Now, line 140 uses the value found by POS to GOTO the appropriate line number to continue the program. The trouble is, an ON GOTO must be able to read a consecutive series of values starting with 1, and if POS gives it a 0, the program will CRASH! That's why the +1 in line 140. If W$ does not end in any of the letters "EFHNSXYZ" then POS=0, +1=1, ON 1 GOTO 320. So let's go to 320 and look at our last string manipulator, the ampersand, "&", "and sign", or in computerese, "concatenation". All it does is to join two strings into one. PL$ is our variable name for the plural form of W$, and in this case it consists of W$ with an "S" tacked onto the end.

If W$ ends in "E", POS=1, +1=2, ON 2 GOTO 150. In line 150, the <> when dealing with strings means "other than" or "is not", and the line means "if the segment of W$ consisting of 2 characters starting with the 2nd character before the last character, is not "IF" then go to 320. In other words, if W$ is not "WIFE" or "KNIFE" or some such, just put an "S" after the word, in 320. Otherwise, in line 160, PL$ (the plural) consists of the segment of W$ starting with the first character and containing the number of characters equal to the length of W$ minus 2, with "VES" tacked on. So, "WIFE" becomes "WIVES", etc. - yes, I know the plural of "FIFE" is not "FIVES", but....! If W$ ends in "F", POS=2, +1=3, on 3 GOTO 180. Remember that Y$ is the last two characters of W$. The coding here had best be the subject of another article, but just read those asterisks as "and" - if Y$ is not "AF" (as in LOAF) and Y$ is not "LF" (as in CALF) and Y$ is not "RF" (as in SCARF) and W$ is not "HOOF" (but it could be ROOF!) then go to 320 to tack an "S" on the end, otherwise drop through to 190 to pick out the segment from the 1st character to the next-to-last and add "VES" to it.

Similarly, in lines 210-220, if words ending in "H" do not end in "CH" or "SH", they take the "S" ending, or "ES". In lines 240-250, if words ending in "N" do not end in "MAN", they take "S"; otherwise, take the segment from the 1st character to the 4th from the end and add "MEN". Words ending in "S", "X" or "Z" are referred to line 270 to add the "ES" ending, and lines 290-300 figure out that words ending in "Y" preceded by a vowel take the "S" ending, otherwise knock off the "Y" and add "IES". I hope that I haven't overlooked some other rule of plural endings here - but anyway, I'm trying to teach you how to program, not how to spell!

SUPERBUG II

By Edgar Bohanna

A little over a year ago, TI sent SUPERBUG to all Users Groups as a public domain program. This was an improved version of the debugger which comes with the Editor/Assembler. I have several other debugging tools including BUGOUT which is a commercial product, but the one I use almost exclusively is SUPERBUG. That is, I used to use SUPERBUG until I recently put the finishing touches on SUPERBUG II.

As you may know, SUPERBUG was an unfinished product released "as is" to the TI users community. It had a few bugs and several deficiencies. The most serious bug was the fact that the dump or disassembly to disk did not work. Patches to fix this bug have been published in a number of newsletters and while they fix the major bug, they do not correct any of the other deficiencies.

SUPERBUG II on the other hand fixes the following bugs:

1) The dump and disassembly to disk problem is fixed. My method is similar to the one by Tom Knight which has been widely published.

2) Memory dumps and disassemblies to >FFFE or >FFFF used to roll over to >0000 and keep going. This is fixed so they terminate properly.

3) Some minor errors existed in the disassembler. MPY and DIV formats were not exactly right. Also Shift and CRU instructions did not use a > sign for operand values greater than 9. These bugs are fixed.

4) The program was not re-entrant because some of the data buffers overlaid the initialization code. This was not a problem if you never left SUPERBUG control but if you did, your only choice was to reload it in order to restart it. This oversight has been fixed.

5) The program was difficult to use in an Extended Basic environment. The documentation did not properly explain how to use it and it filled the available Low-Memory space so there was no room left for loading any assembly routines to test. These problems have been fixed plus a faster means of loading is provided.

In addition to fixing the above bugs, SUPERBUG II includes the following enhancements:

1) An O command is added to allow changing the List device. This is handy in case you want to dump to printer then disk or vice versa or if you want to dump to multiple disk files.

2) A J command is added to allow toggling through a choice of screen colors. Five different selections are available.

3) The output device specification is changed from OUTPUT to APPEND. This does not affect printer output, but it does allow you to send a series of dumps to disk without overwriting previous dumps or having to change disk names every time. Disk outputs are also in DIS/VAR 80 format.

4) The C, F, H, K, N, P, >, and . commands from the TI DEBUGGER have been added to SUPERBUG II. The notes in the HELP file with SUPERBUG indicated that they were omitted to save space. SUPERBUG II is still about 200 bytes less than 8K in size so I don't know what their concern was. I did tighten up some code and eliminated some "dead" code that wasn't being used.

5) Several enhancements to the A (disassemble) and D (memory dump to output device) have been added. In the original SUPERBUG, disk outputs did not have addresses or instruction codes (for disassemblies). I added a - terminator which will allow address/instruction omission to either disk or printer while a carriage return termination will include address/instruction output to either device. The A and D commands will now accept a third (optional) value to specify a relocated data dump or disassembly. Let's say you copied a DSR from >4000 to >5FFF into the memory from >C000 to >DFFF; this feature will let you dump or disassemble from >C000 to >DFFF but the printout or disk file will reflect the area it was relocated from, in this case >4000 to >5FFF.

6) Another addition to the A command is termination with : or ; instead of return or minus. These terminations will cause disassembly into DATA statements only with ASCII translation as a comment. This is useful when you run into a large area of DATA or TEXT in a disassembled program. The : terminator includes address and instruction while the ; terminator eliminates them.

7) In addition to the compressed and normal object code versions that are relocatable, a third object file has been generated. This one loads at >6000 and includes a GROM header so that once it is loaded, SUPERBUG II can be selected from the menu screen. To use this file, you need a means of loading into RAM in the cartridge memory space. You can do this with the Morning Star 128K RAM card (no other 128K card that I know of will allow this), SUPER SPACE (a cartridge from Model Masters in California), RAMPORT (a cartridge with RAM from a company in Canada, or SUPER CARTridge (a roll-your-own RAM cartridge as described in Micropendium June and July 1985 issues).

My main motivation in creating SUPERBUG II has been for my own personal use. I am developing some DSRs of my own and wanted better tools for testing them and I wanted better tools for studying existing DSRs on various products. The new features of SUPERBUG II provide these tools plus make it a more useful program. I am also making the program available to other users through the FREEWARE distribution process. I have spent considerable time and effort on this project so I am asking the modest sum of $8.00 (available from THE 99'ERS ASSOCIATION - INCLUDES DISK, MAILER, AND SHIPPING COSTS) from anyone who would like to make a contribution for this effort.

WINCHESTER INSIGHTS
By Edgar Bohmann

In a previous article on Winchester disk drives, I discussed the various components which make up a Winchester (hard disk) system for the TI-99/4A computer. These include the Myarc Personality Card, a Western Digital 1001-05 or 1002-05 Controller Card, an ST-506 compatible Winchester Disk Drive, a case with a power supply, and a software package called the Winchester Utilities.

In the last article, I stated that all of this could be assembled for between $800 and $1000 for a 5 Megabyte version. These prices still seem to be valid. In the May issue of the MICROpendium, they stated that Myarc will sell their Personality Card for $250. This includes the Personality Card, a manual, and the Myarc Winchester Utilities. Myarc apparently does not stock these cards as standard items but builds them as required so delivery may take a week or two. Also Myarc apparently is not interested in selling complete systems so you have a choice of "rolling your own" or contacting Model Masters.

Model Masters is a distributor in California who will assemble a complete Winchester system for you, format the Winchester, store the Utilities on the Winchester, and ship it to you. The cost will be between $800 and $1000 depending on the current prices for the various components. Since demand is not high for Winchesters at the present time, Model Masters does not keep any complete units in stock but will obtain components and assemble them as orders arrive. Since each unit is assembled, tested, and formatted individually, delivery will probably take about 2 weeks. However, the good news about this is that the system is completely tested and formatted so when you get it all you have to do is plug it in and start using it.

Another advantage of the Model Masters assembly is that you will get a copy of Model Master's enhancements to the Winchester Utilities in addition to Myarc's original version. You may then use either version, but I think everyone who has both will want to use Model Masters' version.

The Personality Card has a minimum DSR which provides an interface between the TI-99/4A I/O buss and the Western Digital 1001-05 or 1002-05 Controller Card. It provides compatibility with application programs using what TI calls Level 3 routines for handling file OPEN, CLOSE, PRINT, INPUT, and DELETE operations. However, Level 1 and 2 type accessing (for sector I/O, formatting, directory management, etc.) are handled through a set of external assembly language routines which are part of the Winchester Utility package.

The Winchester Utility package is somewhat like a Disk Manager in that it contains a set of menu driven user interface routines for such activities as formatting the disk; creating, backing up, and restoring directories; copying and renaming files; mapping (viewing a directory) the disk or changing its name; and other similar disk management features. The user interface routines are written in Extended Basic while the actual disk I/O routines which do the work are written in assembly language.

Model Masters has totally rewritten the Extended Basic portion of the Utility package to make it user-friendly and increase its speed. The original version has three main menues (Disk, Directory, and File Management). Each main menu has several commands that may be requested. Each command (including the Master Menues) is requested by typing two characters (such as DM for Directory Management). Each two letter command you type causes another Extended Basic program to be loaded and run. The original version only allowed you to request file manipulation commands if you were in the File Menu. So even when you learn all the commands, you still have to wait until another X-Basic program loads before you can issue the command you want.

To make matters even more confusing, some of the commands in the various menues were identical. Model Masters has improved all this by making ALL commands unique and allowing any command to be entered from any menu. Thus if you are in the Directory Management Menu but want to delete a file, you may type DF which brings in the Delete File program directly rather than having to first type FM, then wait for the File Menu to load before you are ble to type DF.

Model Masters has also improved the speed of the X-Basic programs by reducing their size and utilizing the Pre-Scan control featuresof x-Basic. In addition, Model Masters has added some extra features such as Diskette Cataloging, Multiple File Copying, and Printer Manipulation. The Model Masters menues have the Menu name at the top of the screen, a window of common features in the center of the screen (Catalog Diskette, View or Set Date and Time, Exit Utilities, etc.) and a set of specific commands for the type of menu at the bottom. The cursor waits on the command line at the bottom of the screen for your entry. Once you type a command, the X-Basic program supporting that feature will be loaded.

After the X-Basic program is loaded, you will be prompted for additional information if it is required to execute the command. Once all required information is available, the X-Basic program will CALL LINK to the appropriate assembly language routine to actually do the work. Most commands let you choose between entering another similar command or returning back to the Master Menu. A nice touch in the Model Masters version is each of the Master Menues (and the programs which run under them) have different screen colors.

The Myarc Personality Card has a clock option which is set when you first load the Utilities. The original version requires the clock to be reset it each time you restart the computer (whether or not you turned power off on the expansion box). The Model Masters version only requires that you reset the clock when the expansion box is powered off. If the optional battery is installed on the Personality Card, you only have to reset it when the battery fails, when the year rolls over, or when you change between Daylight Savings and Standard Time.

In the next article we will look at some of the Directory Management features of the Winchester system which help make it the powerful tool that it is (besides the fact that it's 4 to 10 times faster than floppies).

TIPS #25 FROM THE TIGERCUB

TIGERCUB SOFTWARE
156 Collingwood Ave.
Columbus, OH 43213

The entire contents of Tips from the Tigercub Nos. 1 through 14, with more added, are now available as a full disk of 50 programs, routines and files for just $15.00 postpaid! Nuts & Bolts is a diskfull of 100 (that's right, 100!) XBasic utility subprograms in MERGE format, ready for you to merge into your own programs. Contents include 13 type fonts, 14 text display routines, 12 sorts and shuffles, 9 data saving and reading routines, 9 wipes, 8 pauses, 6 music, 2 protection, etc., and now also a tutorial on using subprograms, all for just $19.95 postpaid! And I have about 140 other absolutely original programs in Basic and XBasic at only $3.00 each!(plus $1.50 per order for cassette, packing and postage, or $3.00 for diskette, PPM) I will send you my descriptive catalog for a dollar, which you can then deduct from your first order.

Many of the users groups are taking a summer break, so I thought I would do the same. I'm going to mail out the July and August issues of the Tips in June (imagine, a TI publication AHEAD of schedule!!) and then go fishing. However, if anyone should by any chance decide to send me an order during the summer, they will still get my same-day service. It seems that I had better clear up a few misunderstandings. The "freeware" offers I have mentioned in past Tips are NOT available from me - send your disk and returnable mailer AND RETURN POSTAGE to the author of the program. And, my copyrighted Tigercub Software programs are NOT freeware. They can only be legally obtained by mail order from me - if you copy them from anyone else, you are stealing! As for the programs which I write and publish or distribute without copyright, they are also not Freeware, they are FREE. I don't want to be paid for them, and I don't think anyone else should be paid for them. Some users groups are putting my copyrighted programs, and those of other programmers, in their software library, "for use but not copying" or "for review and evaluation only". Who do you think you're kidding? I know I won't sell any software to members of pirate clubs, so why should I support them?

If you didn't solve the Long Division Puzzle in Tips #24, try dividing 230709 by 835. As for the solution to the Tigercub Challenge, it was right on the same page! Try creating those DATA statements with the LINEWRITER routine. I don't know why it works, but it does. I've been asked to print more information on the "program that writes a program". I don't have room for a detailed account, but here are the basics. If you tried my TOKENLIST routine in Tips #23 you already have a list of the token codes you will need. I won't go into the way that the computer squishes a program line number into only two characters, but you can accomplish it with DEF L$=CHR$(INT(LN/256))&CHR$(LN -256@INT(LN/256)), where LN has been predefined as the value of the line number. If you need to refer to a program line in a statement, as in GOTO 500, use DEF R$=CHR$(201)&CHR$(IN T(RN/256))&CHR$(RN-256@INT(R N/256)), RN being the line number.

To print a statement or command, simply print its token character. For instance, the token for DATA is 147, so you would print CHR$(147). Note that all the punctuation marks used in programming, such as (and +, are also represented by token codes which are NOT the same as their keyboard ASCII value.

To print a variable name, either numeric or string, just enclose it in quotes, "A" or "A$". To print a value, or an unquoted string (as in a DATA statement), or the word which follows a CALL, you must print CHR$(200) followed by a token giving the number of characters to follow, such as CHR$(5) for a 5-character word such as CLEAR, then the value in quotes. For instance, the token for CALL is 157, so CALL CLEAR is CHR$(157)&CHR$ (200)&CHR$(5)&"CLEAR" . You can simplify that by predefining DEF U$(V$)=CHR$(200)&CHR$(LEN(V$))&V$, and then simply print CHR$(157)& U$("CLEAR"). A quoted string is handled in the same way except that it is preceded by token 199 instead of 200, so you can predefine it as DEF Q$(V$)=CHR$(199)&CHR$(LE N(V$))&V$ - the computer will take care of the quote marks. Each program line must end with CHR$(0), and the last record you print must be CHR$(255)&CHR$(255). A MERGE format file is D/V 163, so open the file with OPEN #1:"DSK1.MERGEFILE ",VARIABLE 163.

Don't print more than 163 characters in a record or the computer will blow its mind! You can print multiple-statement XBasic lines. Be sure to use the double-colon token CHR$(130) as the separator, not two of the CHR$(181) colon tokens. Any errors you make will usually not show up until you try to MERGE or use the program you have created. I/O ERROR 25 means that you forgot the final 255 & 255; DATA ERROR or SYNTAX ERROR probably means that you left off a CHR$(0) or gave the wrong count of characters after CHR$(200). Here's a bit of psyche- delic blues - -

```
100 REM - FRANKIE & JOHNNIE
    by Jim Peterson
110 DIM S(12)
120 CALL SCREEN(2)
130 FOR R=1 TO 12
140 CALL COLOR(R+1,1,1)
150 FOR T=R TO^25-R
160 CALL HCHAR(T,R,32+888,34
-2*R)
170 NEXT T
180 NEXT R
190 DATA 262,294,311,330,349
,392,440,494,523,587,40000
200 FOR N=1 TO 11
210 READ S(N)
220 NEXT N
230 FOR J=1 TO 110 STEP 2
240 CALL COLOR(A+1,1,1)
250 READ T,A
260 CALL COLOR(A+1,A+2,A+2)
270 FOR TT=1 TO T
280 CALL SOUND(-999,S(A),0)
290 NEXT TT
300 NEXT J
310 RESTORE 330
320 GOTO 230
330 DATA 2,1,2,2,2,4,2,7,1,1
1,1,7,2,6,4,4,2,1,1,11,13,1
340 DATA 2,1,2,2,2,4,2,7,1,1
1,1,7,2,6,4,4,12,1
350 DATA 1,11,3,1,2,5,2,6,2,
7,2,9,1,11,1,9,2,10,4,7,1,9,
1,11,7,9
360 DATA 4,7,2,8,2,9,1,11,3,
9,1,11,1,9,4,8,2,7,6,6
370 DATA 4,4,1,11,3,4,4,3,16
,2,1,11,4,7,2,6,4,7,4,6,20,1
,8,11
```

You can too have a blank space in your disk filenames! Just use FCTN V for the blank, instead of the space bar. You can even have a diskfull of 10 programs with invisible filenames consisting of 1 to 10 of those FCTN V's. However, those invisible characters can do strange things when you list your disk catalog to a printer. If you want to INPUT a string with leading and/or trailing blanks, just enclose the whole works in quotation marks. Try this little gem of a program.

```
100 INPUT A$ !type TEST
110 PRINT A$;LEN(A$)
120 INPUT A$ !type " TEST "
130 PRINT A$;LEN(A$)
140 GOTO 100 ! you can even
input a blank string of 136
characters
```

I really shouldn't tell you this, but if you want to make it difficult for someone to LIST your program, just insert a garbage line, every 5th line or so until you run out of memory, consisting of REM followed by 4 or 5 lines of random characters typed with the CTRL key held down. Here's a program that can actually read your mind!

```
100 CALL CLEAR
110 PRINT "TIGERCUB MIND REA
DER PROGRAM": :
120 PRINT "I'll bet you a do
llar I can guess what you ar
e thinking.": :
130 GOSUB 440
140 PRINT "And I'll bet  ano
ther dollar I can tell if wh
at you are   thinking is cor
rect.": :
150 GOSUB 440
160 PRINT "And I'll bet anot
her dollar I'm right BOTH ti
mes.": :
170 GOSUB 440
180 PRINT "And I'll bet one
more dollar I can guess what
 you'll be thinking a minute
 from  now.": :
190 GOSUB 440
200 PRINT "OK....": :
210 GOSUB 480
220 PRINT "You're thinking t
hat a compu-ter can't possib
ly know    what you are thin
king.............right?": :
230 GOSUB 480
240 PRINT "So I told you wha
t you were":"thinking.......
250 GOSUB 480
260 PRINT "You owe me a buck
270 GOSUB 480
280 PRINT "And you're absolu
tely       right..I can't re
ad your    aind.": :
290 GOSUB 480
300 PRINT "So I told you cor
rectly that":"what you were
thinking was":"correct......
right?": :
310 GOSUB 480
320 PRINT "You owe me anothe
r buck.": :
330 GOSUB 480
340 PRINT "So I was right BO
TH times...right?": :
350 GOSUB 480
360 PRINT "That makes three
bucks you  owe me.": :
370 GOSUB 480
380 PRINT "And now it's a mi
nute later":"and you're thin
king you've":"been played fo
r a sucker....":"...right?":
 :
390 GOSUB 480
400 PRINT "...so you owe me
four bucks.": :
410 GOSUB 480
420 PRINT "NEVER NEVER bet a
gainst a   computer!! "
430 END
440 PRINT "Want to bet? Type
 Y(yes)": :
460 IF (ST=0)+(K<>89)THEN 45
0
480 FOR D=1 TO 800
490 NEXT D
500 RETURN
```

Since the manual doesn't mention it, some folks don't know that you can use IMAGE and PRINT USING for output to the printer. Try this:

```
100 OPEN #1:"PIO"
110 INPUT "NAME? ":N$
120 INPUT "AMOUNT? ":A
130 PRINT #1,USING "########
##########              ####
.##":N$,A
140 GOTO 110
```

Of course, you could also add a line:

```
105 IMAGE "##################
####.##"
```

And change line 130 to 130 PRINT #1,USING 105:N$,A

Attention, assembly programmers! Fred Hawkins of the Lehigh U6 is trying to coordinate a project of documenting the operating system by breaking the console ROM down to pages of 256 bytes so that each individual or group can work on just one page. Only those who participate will share in the results! All this is far beyond me, but if you want in, send an SASE and a SSSD disk with return postage and mailer to Fred Hawkins, 1020 N 6th St, Allentown PA 18102 - soon!

If you have a program on disk which is so long that you must type CALL FILES(1) before you can load it, add several program lines to it consisting of REM and any key you want to hold down for 5 lines. Then SAVE it back to the disk; it will now be in INT/VAR 254 format and will load without CALL FILES(1). If you then need sometime to make a casette copy, just delete those lines and SAVE it back to disk again. If a program loads, but gives you a MEMORY FULL IN LINE has used up all available memory while reading DATA into arrays or performing other internal calculations. If it runs for some time and then gives you the MEMORY FULL message, it is because you have repeatedly jumped out of a FOR...NEXT loop with an IF...THEN...GOTO before the loop is completed. This rarely happen but it can, especially when you repeatedly jump out of the innermost of several nested loops.

MEMORY FULL Jim Peterson

HINTS 'N TIPS

PEEKS & POKES

****

THESE ARE ALL OF THE PEEKS & POKES THAT WE HAVE COME ACROSS FOR USE WITH X-BASIC AND 32K MEMORY EXPANSION (BE SURE TO DO A "CALL INIT"). THE P & Q VARIABLES ARE USED FOR "PEEK" - THE NUMBERS ARE FOR "POKE" OR "LOAD". IF YOU KNOW OF ANY OTHERS PLEASE LET ME KNOW AND WE WILL ADD THEM IN. - EDITOR

****

ADDRESS , VALUE(S)	MEANING IN EXTENDED BASIC
	CALL VERSION(X) IF X=100 100= NEWEST VERSION OF X/B CART
8192 , P	USE (PEEK,P) IF P<> 70 OR <>121 THEN DO A CALL INIT
8194 ,	FIRST FREE ADDRESS IN LOW MEMORY
8196 ,	LAST FREE ADDRESS IN LOW MEMORY
-28672 , P	P=0 SPEECH NOT ATTACHED P=96 OR P=255 SPEECH IS ATTACHED
-31572 , 0 TO 255	VARY KEYBOARD RESPONSE
-31740 , P , Q	PUT IN DIFFERENT TO CHANGE BEEPS,WARNINGS, ETC
, 192	NO AUTOMATIC SPRITE MOTION OR SOUND
, 224	NORMAL OPERATION
, 225	MAGNIFIED SPRITES
, 226	DOUBLE SIZE SPRITES
, 227	MAGNIFIED & DOUBLE SIZED SPRITES
, 232	MULTICOLOR MODE (48 BY 64 SQUARES)
-31794 , P	TIMER FOR CALL SOUND (COUNTS FROM 255 TO 0)
-31804 , X , Y	RETURN TO THE TITLE SCREEN (USE "PEEK (2,X,Y)")
, P	CHANGE THE CURSOR FLASH RATE (0 TO 255)
-31806 , 0	NORMAL OPERATION
, 16	DISABLE QUIT KEY (FCTN =)
, 32	DISABLE SOUND (USE NE6 DUR FOR CONTINOUS SOUND)
, 48	DISABLE SOUND & QUIT KEY
, 64	DISABLE AUTO SPRITE MOTION
, 80	DISABLE SPRITES & QUIT KEY
, 9	DISABLE SPRITES AND SOUND
, 128	DISABLE ALL THREE
-31808 , P , Q	DOUBLE RANDOM NUMBERS (0 TO 255) NEED "RANDOMIZE"
-31860 , 4	GO FROM EX-BASIC TO CONSOLE BASIC (NEED "NEW")
, 8	AUTO RUN OF DSK1.LOAD
-31866 , P , Q	END OF CPU PROGRAM ADDRESS (P*256+Q)
-31868 , 0	NO "RUN" OR "LIST" AFTER "BREAK" IS USED
, 0 , 0	TURNS OFF THE 32K MEMORY EXPANSION
, 255 , 231	TURNS ON THE 32K MEMORY EXPANSION
-31873 , 3 TO 30	SCREEN COLUMN TO START AT WITH A "PRINT"
-31877 , P	P&32 = SPRITE COINCIDENCE P&64 = 5 SPRITES ON A LINE
-31878 , P	HIGHEST NUMBER SPRITE IN MOTION (0 STOPS ALL)
-31879 , P	TIMER FOR VDP INTERRUPTS EVERY 1/60 OF A SEC (0 TOP 255)
-31880 , P	RANDOM NUMBER (0 TO 99) NEED "RANDOMIZE"
-31884 , 0 TO 5	CHANGE KEYBOARD MODE (LIKE "CALL KEY(K,...)")
-31888 , 63 , 255	DISABLE ALL DISK DRIVES (USE "NEW" TO FREE MEMORY)
, 55 , 215	ENABLE ALL DISK DRIVES (USE "NEW" TO FREE DRIVES)
-31931 , 0	UNPROTECT X-B PROTECTION
, 2	SET "ON WARNING NEXT" COMMAND
, 4	SET "ON WARNING STOP" COMMAND
, 14	SET "UNTRACE" COMMAND
, 15	SET "UNTRACE" COMMAND & "NUM" COMMAND
, 16	SET "TRACE" COMMAND
, 64	SET "ON BREAK NEXT" COMMAND

, 128	PROTECT X/B PROGRAM
-31952 , P	PEEK P=55 THEN 32K EXPANSION MEMORY IS OFF <>55 MEANS6ON
-31962 , 32	RETURN TO THE TITLE SCREEN
, 255	RESTART X/B W/DSK1.LOAD
-31974 , P , 0	END OF VDP STACK ADDRESS (P*256+0)
-32112 , 8	SEARCHES DISK FOR ?
-32114 , 2	RANDOM GARBAGE
, 13	SCREEN GOES WILD
, 119	PRODUCE LINES
-32116 , 2	RANDOM CHARACTERS ON SCREEN
, 4	GO FROM X/BASIC TO BASIC

ADDRESS , VALUE(S)	MEANING IN EXTENDED BASIC
-32187 , 0	UNPROTECT XB PROGRAM
, 2	SET "ON WARNING NEXT" COMMAND
, 4	SET "ON WARNING STOP" COMMAND
, 9	SET 0 LINE NUMBER
, 14	SET "UNTRACE" COMMAND
, 15	SET "UNTRACE" COMMAND & "NUM" COMMAND
, 16	SET "TRACE" COMMAND
, 64	SET "ON BREAK NEXT" COMMAND
, 128	PROTECT XB PROGRAM
-32188 , 1	CHANGE COLOR AND RECEIVE SYNTAX ERROR
, 127	CHANGE COLOR AND RECEIVE BREAKPOINT
-32630 , 128	RESET TO TITLE SCREEN
-32699 , 0	UNPROTECT XB PROGRAM
, 2	SET "ON WARNING NEXT" COMMAND
, 4	SET "ON WARNING STOP" COMMAND
, 14	SET "UNTRACE" COMMAND
, 15	SET "UNTRACE" & "NUM" COMMAND
, 16	SET "TRACE" COMMAND
, 64	SET "ON BREAK NEXT"
, 128	PROTECT XB PROGRAM
-32700 , 0	CLEARS CREEN FOR AN INSTANT
-32729 , 0	RUN "DSK1.LOAD"
-32730 , 32	RESET TO TITLE SCREEN
-32961 , 51	RESET TO TITLE SCREEN
, 149	SETS "ON BREAK GOTO" LOCKS SYSTEM

THE FOLLOWING LOADS REQUIRE E/A OR MM

ADDRESS , VALUE(S)	MEANING
784 , P	USE POKEV(784,P) (WHERE P IS 16 TO 31) CHANGES BACKGROUND COLOR OF CURSOR
-24574 , 8	I THINK THIS ALLOWS MINI-MEM TO USE THE 24K FOR STORAGE
-30945 , 0	WHITE EDGES
-32272 , 0 , "" ,	-30945 , 0) WILL PUT YOU IN TEXT MODE
-32766 , 0	BIT MAP MODE
-32768 , 0	GRAPHICS (NORMAL MODE)
-32280 , 0	MULTI-COLOR MODE
-32352 , 107	WILL BLANK THE SCREEN, ANY KEY PRESS WILL RESTORE

$ PASCAL LOADS

14586 , 0 , 0	THIS ALLOWS YOU TO DO A "RUN-TIME WARM START" FROM PASCAL TO BASIC

THAT'S ALL FOR THIS MONTH, FOLKS!!

BULK RATE
U. S. POSTAGE
PAID
Permit No. 834
Bakersfield, Calif.

SUBSCRIPTION FORM
FOR "THE NATIONAL NINTY-NINER"
PUBLISHED BY THE 99'ERS ASSOCIATION

NAME: _____ DATE: _____

ADDRESS: _____

CITY: _____ STATE _____ ZIP _____

SUBSCRIPTION TYPE	AMOUNT	CHOICE
THIRD CLASS - BULK RATE	$12.00	_____
FIRST CLASS - US AND CANADA	$17.00	_____
FIRST CLASS - OVERSEAS	$22.00	_____

PLEASE MAIL CK./M.O. FOR SUBSCRIPTION CHOICE SELECTED ABOVE TO:

THE 99'ERS ASSOCIATION
ATTN: LUCI VEITH
3535 SO. H ST., #93
BAKERSFIELD, CALIF. 93304