CLEVELAND AREA 99-4A USERS GROUPS NEWSLETTER

DECEMBER, 1986

****

OFFICERS	SOLON	NORTHCOAST	GOLDEN CRESCENT	TI-CHIPS
PRESIDENT	STEVE WEINKAMER (1-274-2544)	RON MINADEO (943-2791)	CHUCK MARENO (324-4389)	RICH POLIVKA (238-3971)
VICE PRESIDENT	WALT RYDER (921-8223)	MARTIN SMOLEY (257-1661)	DICK BURGER (967-1317)	RUSS SHIMANDLE 1-887-5330
TREASURER	FRANK JENKINS (283-8526)	JIM MEKEEL (284-3179)	DALE SCHIEFERSTEIN (988-5150)	DAN KELLER (225-0658)
SECRETARY	RUSS OGDEN (1-626-2622)	JOHN BLACKMER (257-2672)	HAROLD SKIPWORTH (631-4529)	JAN FEDOR (524-6192)
LIBRARIANS	KIM JONES(1-274-8019)	DEANNA SHERIDAN (333-5986)	BOB BARTO (1-282-7037)	MARK McCAULEY (235-8888)
	LADD PATA (662-9253)			

BULLETIN BOARD----------(24 HOURS) (216)(944-1072)

****

EDITORIAL COMMENTS

This issue marks the end of the first year of publication under the direction of the Northcoast group. We have expanded the newsletter by 2 pages, plus the use of the condensed print, expands it another couple of pages. Compared with the old, this newsletter gives you about 16 pages worth of information.

Special thanks to those who have faithfully supported us the past year either by attending meetings or writing articles: Tom Nellis, Bruce Young, Frank Jenkins, Ken Sladyszewski, Marty Smoley, Steve Weinkhemer, Russ Shimandle, Mark McCauley, Chuck Mareno, Jim Mekeel, Les Kaas, Les Israel, Dave Talan.

Let's see how much better we can be next year. It has been great to see our club's home-grown articles appear in other newsletters throughout the country. If I could get some sub-editors to take charge of departments such as programming, hardware, reviews, etc., we could feature these departments on a consistant basis. As it is, I go with anything local and then fill in with what's out there, and sometimes we don't have too much choice as to what to pull out for our use and don't have the variety I would like to have in an issue.

Got a flyer from Millers Graphics this past week, and he says that by January, 87, we will all be able to use IBM software on our machine, add real IBM cards, run Lotus, Wordstar, etc. This will really be something if it comes true.

Our Tom Nellis has an article in the November issue of Metal Stamping, an industry publication, concerning the quality-control software his company is using in the auto parts area.

A newsletter listing publications available for the TI99 mentioned Nuts and Volts, P. O. Box 1111, Placenia, CA 92760. Subscription $10/yr, $35/life. Says it is a recycler kind of newspaper..almost all classified ads with some really neat electronic and computer equipment, including a fair number of ads listing 99 consoles and peripherals.

You can get a free sample of MICROpendium by calling 512-2255-1512. If they can get 8,000 more subscribers, they can add several pages to the magazine. You can also send a post card to P.O. Box 1343, Round Rock, TX 78680. If you enjoyed Regena's columns in either the old 99er Magazine, Enthusiast, or Compute!, you will look forward to her first article in MICROpendium starting with the January, 1987 issue.

Issue 84 of Genial Traveler arrived this week, but I haven't had a chance to put it in the machine yet.

TI-Talk is up and running on the FREE-NET and seems to be getting lots of use in a short time. The entire FREE-NET is a one-of-a-kind experiment never done before anywhere, and we should be excited about being able to be a part of it.

Have a HAPPY HOLIDAY SEASON and we'll see you next year. I expect everyone of you to make a New Year's resolution to give at least 1 presentation at a group meeting, or write an article for the newsletter, or serve on a committee, or donate software to the library and we will have one of the best years ever!!!!

Look inside for...A HA HA about Noah, A review of the Diagnostic Disks received From TI (which also included an updated list of user groups); A tutorial in c99 programming; How to give your Horizon Ramdisk 256K; a report on the Chicago FAIRE by Art Byers; Odds and ends of programming hints; a tutorial in Basic programming; and a review of Pro-Scan IT! from Asgard.

EXECUTIVE NOTES - TI-CHIPS

A hearty thanks goes to all who made presentations at the November meeting. Matt Andel gave a LOGO II graphics demonstration; Less Kee dissected another Basic program, and Terry Vacha demonstrated Expanded Graphics Basic. Mark McCauley also made a music and graphics presentation. Thanks to the Thalners of Edu-Comp for the loan of the Expanded Graphics program for Terry's demo.

The December meeting will be short, as a Christmas party will follow with refreshments. All are invited to bring a favorite snack. We will also have a "Swap and Sell", so be sure alto to bring anything you wish to sell.

Election of officers will be in January. Anyone interested in an office should submit names to Russ Shisandle or any officer. Please remember that we need a new membership chairman, since the Treasurer is only filling this position on a temporary basis.

Congratulations to Otto Hagele for winning the drawing for the computer paper.

We look forward to seeing you next month. Ed Kennelly will give a presentation on Extended Basic Graphics; Matt Andel will present LOGO II, Part II, and the Thalners will have a RAM disk plus a working Home Control Unit. There will be another raffle and maybe some "surprises". See you on December 20 at 10:00 a.m. in the North Royalton Public Library.

JAN FEDOR

EXECUTIVE NOTES - GOLDEN CRESCENT

The next meeting of the Golden Crescent UG will be on Thursday, December 18, at "The Computer Advantage" at 7:30 p.m., 920 Archister Dr., Amherst, OH, PH 960-1312. The store is located off 254 near K-Mart.

The last meeting was held on Thurs, November 20 at The Computer Advantage. Thanks to Dick Burger for the use of his classroom for our meeting. Dick has just moved his opEration from Vermilion to Amhert. His new store is very impressive. Dick has spent a lot of time and effort to have the latest and best in stock. I urge all club members to stop in this Christmas season and say "Hello".

Thanks to Tom Mellis, we are now off to a good start on our A/L SIG. Tom was kind enough to supply our group with a video, several disks and some hard copy on A/L. Any member who would be interested in this SIG, contact Duayne Hughes at 949-6848. Also, at the last meeting we purchased 25 disks on which we will place the latest additions to the club library. The library will be at the Computer Advantage.

In closing, I would like to wish all our members, past and present, a safe and Happy Holiday Season. I know the club officers join me in this wish. So, stop in and see us on December 18. We will have a very interesting meeting.

CHUCK MARINO

THE ARK AND THE COMPUTER
PUGET SOUND 99ERS, JULY, 1986

And the Lord said unto Noah: "Where is the Ark which I have commanded thee to build?

And Noah said unto the Lord: "Verily, I have had three carpenters off ill. The gopher wood supplier hath let me down...yea, even though the gopher wood hath been on order nigh upon 12 months. What can I do, Lord?"

And God said unto Noah: "I want that Ark finished even after seven days and seven nights!"

And Noah said, "It will be so."

And it was not so. And the Lord said unto Noah: "What seeseth to be the trouble this time?"

And Noah said unto the Lord: "Mine subcontractor hath gone, alas, bankrumpt. The pitch which thou hath commandest me to put on the outside and the inside of the Ark hath not arrived. The pipefitter hath gone on strike. Shem, my son, who helpeth me on the Ark side of the business, hath formed a rock group with his brothers Ham and Japeth. The canvas, although on hand, is not the right color. Lord, I am undone."

And the Lord grew angry and said: "And what about the animals, the male and the female of every sort that I hath ordered to come unto thee to keep their seed alive upon the face of the earth?"

And Noah said: "They hath been delivered unto the wrong address but should arriveth on Friday."

And the Lord said: "how about the unicorn and the fowls of the air by sevens?"

And Noah wrung his hands and wept aloud, saying: "Lord, unicorns are a discontinued line, one canst not get them for love or money. And fowls of the air are sold only in half-dozens...the peacocks even then, are on back order for weeks to come. Lord, Thou knowest how it is."

And the Lord said: "Rest, my son. In my wisdom I should have known this would follow when I listened to Satan and installed a computer."

FOR SALE: 300-BAUD Volksmodem by Anchor Automation with cable for TI. Owner recently purchased IBM compatible computer and cable is incompatible. Using this opportunity (excuse) to get 1200-baud.

$3600 complete
333-3984 - Deanna Sheridan

THE PARENT COMES THROUGH

By Jim Nakael
Northcoast 99ers
November 15, 1986

The 99/4A may be an orphan, but Texas Instruments has not disinherited the users groups yet. After several months of waiting, we received the Software Testing System that TI announced it would release. This release consists of two disks and approximately 24 pages of documentation. One disk is used with Extended Basic and the other is used with the Mini Memory module. Per TI's cover letter, the functions of the two disks are the same. This month we shall review the XB version.

Cataloging reveals a LOAD program with the following menu appearing:

```
 0...P/CODE TEST
 1...EXPANSION BOX TEST
 2...IMPACT SERIAL TEST
 3...IMPACT PARALLEL TEST
 4...SPEECH TEST
 5...THERMAL PRINTER TEST
 6...DISK EXERCISER
 7...RS232 TEST
 8...MODEM TEST
 9...CATALOG TEST
10...RS232/1&2 TEST
11...RS232/3&4 TEST
```

P/CODE TEST Tests the two ROM's and eight GROM's of the P-Code card individually. Since I do not have this card, I cannot tell you more about this test.

EXPANSION BOX TEST

Simply states good or bad unit.

IMPACT SERIAL TEST

Transmitting only at 300 baud, this program sends all the ASCII characters, type sizes, type densities, sounds the bell, horizontal tab, vertical tab and follows with a graphics test. This is useful in testing printers (TI, Epson, Star, and compatibles) and the first RS232 port RTI function. Note - the test does not send out all the available characters that are non-ASCII. If you want to test for these, use a program from the Northcoast library called GEMINI DEMO which can be found on disk 66.

IMPACT PARALLEL PRINT TEST

Same as above, except uses the parallel port.

SPEECH TEST

This one speaks the entire vocabulary contained in the speech synthesizer and prints the words to the screen. Takes about 4 minutes to speak all the test.

THERMAL PRINTER TEST

Prints rows of characters to test the printer.

DISK EXERCISER

The documentation is not very good on this one. The screen comes up with:

```
step in <
step out
seek 16
drive sel 1
Restore
Adjust 00
write xx
side select 0
Track Reg xx
write protect x
index pulse
track 00 x
read data
```

then PRESS ANY KEY TO BEGIN

Pressing any key starts the best (better use a blank disk until we know more!) Drive #1 turns on and the read data numbers start counting up.

I left this on for about four minutes with no other action occurring. Pressing different key combinations does not change anything. Quit is disabled - you must turn off the console to get out of this one. Using Disk Helper 1, I cannot see that anything has been written to the disk. The program is in assembly, and I see that there is a KEY SCAN routine but no documentation. Perhaps one of the smart assembly programmers in our group can help us out?

It could be that this program would be useful in head cleaning since the drive rotates constantly.

RS232 TEST

This test requires an adapter to be built and will report errors in the buss signals, ROM, CRU registers 1302 through 130E, DTR inputs at the UART's and the buffers.

MODEM TEST

Requires a modem with a test switch. Most modems do not ave this. All the test switch does is to loop back to the signal. This test is most commonly used with accoustic coupler type modems.

CATALOG TEST

Simply catalogs a disk in drive #1.

RS232/1&2 TEST

Again, requires an adapter cable like the previous RS232 test. Without the cable, the machine tells you "BAD ROM".

RS232/34 TEST

Same as above.

For the most part, this disk provides some useful tests in trouble shooting your hardware. However, in many areas, the documentation leaves a lot to be desired. Perhaps TI will see fit to distribute an update to the current documentation to make these test programs more usable.

PROGRAMMING IN C
AN INTRODUCTION - PART 1
FROM CENTRAL TEXAS 99-4A USERS GROUP

Before we get started, I'd like to tell you how I intend to describe elements of the C language in these articles. Most C programs are written entirely in lower case. I'll do this too. However, when you see something in these articles that is in upper case, I'll be indicating something of a general nature that goes in a C program at this point. A blank to fill in, so to speak.

To demonstrate (and to start), let's look at the general structure of a C program.

```
VARIABLE DECLARATIONS
main()
{VARIABLE DECLARATIONS
 STATEMENTS
}
```

VARIABLE DECLARATIONS indicates where variables are declared in a program. STATEMENTS indicate where the executable statements of the program are placed.

In C, as with many other languages, it is necessary to declare the existance of all the variables in the program before the executable statements. This helps produce faster running programs, as well as helping the programmer to detect misspelled variable names.

In C there are three basic types of variables: Integers (which are 16-bit values), characters (which are 8-bit values), and pointers (which we aren't going to discuss now). One should remember that while integer and character variables are signed (they can be negative), they are the counting numbers and are not floating point numbers. That is, an integer can have the value of 1 or 2, but not 1.5. If you want a program that needs floating point numbers, use BASIC. For most programs, integers are all we really need, and calculating with them can be very fast.

To declare integers in C, use the following statement:

```
int VARIABLE NAME 1, VARIABLE NAME 2, ..., VARIABLE NAME N;
```

Here is an example:

```
int a,b,c;
```

This declares three variables a,b, and c and sets aside memory for them.

Notice that multiple variables can be declared with this statement by separating them with commas. C knows that it has reached the end of the list when it finds the semicolon.

To declare a character, use the following statement:

```
char VARIABLE NAME 1, ..., VARIABLE NAME N;
```

An example would be:

```
char x,y,z;
```

This declares variables x,y, and z and sets aside space for one character for each. That's right, "char" variables can hold only one 8-bit value.

That isn't very useful in most programs. Usually we want to have strings of characters. To do this, we simply declare an array of characters:

```
char VARIABLE NAME [LENGTH];
```

Here are a couple of examples:

```
char w[10];
int d[5];
```

This also shows how to declare an array of integers. The array w has space for 10 characters set up for it, and d has space for 5 integers.

That should be enough about variables for now, let's move on to statements.

By far the most common statement in a C program is the assignment statement. It looks like assignment statements in most other languages:

```
VARIABLE NAME = EXPRESSION;
```

Where EXPRESSION is a calculation to be performed. Here is an example:

```
a = b + 1;
```

In this example, the value of b has 1 added to it, and the result is placed in variable a. Note the semicolon. In C the semicolon is used to indicate the end of all statements.

The operation on the right of the equal sign is called an expression. In C, expressions can be very complex and are very flexible. Some of the operations available are: multiplication (*), integer division (/), addition (+), subtraction (-). Notice the integer division. This means that if you divide 3 by 2, the result is not 1.5, but 1. If you want the remainder of a division, you use a different operation called modula (%):

```
b = 5 % 2;
```

Here, the result of 1 is placed in the variable b.

In C, if an expression contains multiple operations, they are performed from the left to the right, with multiplications and divisions being performed first, followed by additions and subtractions. Parenthesis may be used to change this ordering.

Although most people don't think of them this way, C contains some other operations: less than (<), less than or

equal (<=), greater than (>), greater than or equal (>=), not equal (!=) and equal (==). No, that last one is not a typo. "==" is used to distinguish the comparison operation from "=" that is used for assignment.

When a comparison is made between values using the above operators, the result that is returned is 0 for false and 1 for true. To C, an expression that evaluates to 0 can be considered to be "false", and one that evaluates to anything else is considered to be "true". The importance of this will become more apparent in a few moments.

There are other operators that C supports in expressions, but we'll save those for another class. Next, let's turn our attention to the variables and constants of the expressions.

Normal variables are used as you would expect. Just place them in the expression, and the current value of the variable will be used for the calculation. If a character variable is used in an expression with integer variables, then the character variable is converted to a 16-bit value and the calculation proceeds. If an integer expression is assigned to a character variable, then the result of the expression is truncated to 8 bits and placed in the variable.

If we simply wish to use a constant in an expression, that is allowed also. Numeric constants like 1, 2, 100, are treated like decimal integers. If desired, character constants can be used by placing them in single quotes. '0' is a character constant that is the ASCII value of the character 0.

References to arrays are straight forward. Simply use the name of the array and put the subscript in brackets ([]) following it. One dimensional arrays only are allowed in Small C. The first element in the array has subscript 0, and the declaration of the array reserves space for requested number of elements. Thus if 10 elements of the array are reserved, then the valid subscripts range from 0 to 9.

Actually, that is a bit misleading. In C, subscripts are not checked while the program is running. Thus, if an array is declared with 10 elements, and the reference is to the 50th element, then C will not stop the program. Instead, some memory location past the end of the array will be used.

Another thing that can appear in expressions is a function. This is the name of some other subroutine in the C program. When the name is encountered in the expression, evaluation of the expression is suspended, and the function's code executed. The function's code will generate a value when it is through, and this value will be what is used to complete the evaluation of the expression. Functions are a topic in itself, and we'll discuss them in another class too.

Let's turn our attention now to other statements in C, and the "i" statement is a good place to start, it looks like this:

```
if (EXPRESSION)
   STATEMENT;
```

This says that if the EXPRESSION evaluates to true, then the STATEMENT is executed. Simple enough, let's look at a more complex "if":

```
if (EXPRESSION)
   STATEMENT-1;
 else
   STATEMENT-2;
```

The "if" statement can also dictate the action to take if the statement is false. In this case STATEMENT-2 will be executed.

Finally, let's look at the "while" statement. This is one way of creating loops in C.

```
while (EXPRESSION)
   STATEMENT;
```

This says that while the EXPRESSION evaluates to true, the STATEMENT will be executed repeatedly. (The STATEMENT will need to make sure that the expression will eventually evaluate to false.)

Now in the above examples, it was implied that only one statement would be executed, and this is true. However, if you wish to place multiple statements in the "if" or "while" statements, this can be done with the compound statement. This is simply enclosing several statements in braces ({}). Here is an example:

```
if (a < b)
   {a = b;
    b = b +1;
   }
```

This should be enough of an introduction to the C language to write a program. Let's take a classic, generating prime numbers between 3 and 100.

```
#asm
 REF PRINTF
#endasm


main()
{int i, j, r;
 i = 3;
 while (i < 100)
   {j = 2;
    r = i % j;
    while (r != 0 & (j * j < i))
      {j = j + 2;
       r = i % j;
      };
    if (r != 0)
       printf("%d is prime/n", i);
    i = i + 2;
   };
}
```

A few notes about this program. First, it's not the

most efficient implementation of a prime number generator because that's not what I'm trying to demonstrate here. Second, the "Basa REF Bendasa" and "printf" are features of the language that I haven't described yet, but they are necessary to let the progras output it's results. Third, the "&" likewise hasn't been described yet, but in this case, it can be read as "and".

In this progras, the variables, i, j, and r are defined. The variable i contains the current value that we are trying to verify as prime. The variable j contains the current value that we'll divide into i and see if it will divide evenly and thus prove that i isn't prime, and r contains the remainder of that division.

The loop starts with 3 as the first value to check and continues until i is greater than 100. The variable j starts with 3 as the first divisor to try and the first remainder is calculated. The progras then loops until a divisor that evenly divides in i is found, or j becomes greater than the square root of i. If this happens, then i is proved to be a prime number.

If i is prime, that fact is output, and the next odd number is taken in an attempt to prove that it is prime.

C programs can be entered using the standard Assembler/Editor text editor. I recommend saving them with a name like DSK1.PRIME:C. Appending the :C is a way of telling that it is a C progras later when you are looking at a directory of the disk. It also will be useful when we are compiling the progras.

When we execute a BASIC progras, there is a progras running in the computer called an interpreter that scans the BASIC progras and performs the operations that the statements in the BASIC progras request. This is very efficient as far as saving memory, but is slow.

The C compiler, on the other hand, is a progras that executes in the computer and reads the statements of a C progras and produces an assembly language progras that will perform the requested operations. Since the progras will be in assembly language, it will execute very quickly as opposed to the BASIC progras.

To run the Small C compiler, use option 5 from the assembler editor cartridge, RUN PROGRAM FILE. When it requests a file name, enter "DSKx.C99C", depending on which disk drive the Small C diskette is in. After the compiler is loaded, it will ask "Include c-text?". Respond with N for now. It will also ask "Inline push code?". Respond with N for this too.

It will then ask for an input file. Give it the name of the file that you saved the progras in (DSK1.PRIME:C). It will then request an output file. I think that it is best to give it a name similar to the original name, except append a :I to it. (DSK1.PRIME:I). Here :I stands for intermediate. This file can be deleted later.

If there are errors in the C progras, then the compiler will complain. It will display the line where it believes there to be a problem and then will point to the place where the problem occurred with a caret. Finally, it will give an error message describing the problem. You'll have to press <ENTER> to get the compiler to continue. Make a note of the line and the problem so that you can correct the problem later. If there doesn't appear to be anything wrong with the line, then the problem probably occurred on the previous line. I can tell you now that the most frequently occurring problem in C programs is forgetting the ; after statements, and the compiler never notices this until the next line. Simply reedit the progras after the compiler completes to fix the problems and then recompile.

When the compiler executes and finds no problems, it will produce an assembly language source progras in the output file. This will have to be run through the /4A's assembler to produce the final executable progras. So, load the assembler, specify the compiler's output file as the source file (DSK1.PRIME:I), and then specify an object file. Here I recommend appending :O to (DSK1.PRIME:O) to distinguish it from the other files. Don't bother producing a listing file and no options are necessary.

If there were errors assembling, reexamine the C progras for errors. It may be necessary to look at the assembly language progras for clues.

Assuming all went well, you are now ready to execute the progras. Use option 3, LOAD AND RUN, and enter the name of the assembler object file (DSK1.PRIME:O) as the first file to load. Next load the C support routines. They are on the C diskette and are called DSKx.CSUP. Finally, for the PRIME progras to run, you must load DSKx.PRINTF.

After the last object file is loaded, press <ENTER> at the FILE NAME prompt and you'll be prompted for the progras name. Type the letters START but don't press <ENTER> yet!

Take a deep breath, buckler your seat belts and now press <ENTER>.

Zoooooooooooooo!

The prime numbers between 3 and 100 will be generated faster than you can see them!

In the next class (article), we'll try to cover less ground with more details. We'll discuss functions, input, and output.

Till then, type in the prime number progras above and at least go through the steps of compiling and executing it.

Mike Schultz

PS: Holy Moley! There's a bug in the example progras. If you look real quick while it is running, you'll notice that it doesn't generate 3 as a random number. See if you can fix it.

MS

HORIZON RAMDISK 256K EXPANSION PROJECT

By Edward A. Hallett
SOUTHWEST NINETY-NINERS, October, 1986

The HORIZON RAMDISK is available in 90K SSSD (360 Sectors) and 180K DSSD (720 Sectors). This project expands the size to 256K (976 Sectors) for an increase in storage capacity of 64K (256 Sectors) or 35.5%.

This increase is accomplished by adding one 74LS154 (4 to 16 DECODER), one 74LS02 (NOR GATE), and eight 8K 6264LP-15 STATIC RAM chips, removing one 74LS138 (3 to 8 DECODER) chip, and modifying the DSR CODE to recognize the existance of the added memory. The original HORIZON RAMDISK CIRCUIT does not fully decode on of the five memory address lines from U9 limiting it to 180K. By fully decoding this line, we pick up eight more CHIP SELECT SIGNALS bringing us up to 256K (976 Sectors). This utilizes the original design to its fullest potential with only a few SIMPLE MODIFICATIONS.

The HORIZON SOURCE CODE VER_03 was used in this project, but modifications to other versions should be very similar. (Note: VER_04 arrived as this was going to press. All modifications were identical, except MEMTEST. Use the modified MEMTEST from VER_03 to check the memory added by this EXPANSION PROJECT.

::
CAUTION: THIS MODIFICATION IS UNDERTAKEN AT YOUR OWN RISK
AND MAY VOID YOUR HORIZON WARRANTY.
::
CAUTION: REMOVE THE NICAD BATTERIES FROM THE RAMDISK
BEFORE STARTING. USE CARE WHEN HANDLING THE RAM CHIPS
TO AVOID DAMAGE FROM STATIC
::

1. Remove U1, the original 3 to 8 DECODER CHIP, from its socket and DISCARD.
2. Remove the EIGHT PIGGYBACKED PAIRS OF 8K RAM CHIPS from their sockets U3-U6 and U12-U16.
3. Remove U2, the original 4 to 16 DECODER, from its socket.
4. Remove U10, the original NOR GATE, from its socket.
5. Install a THIRD ADDITIONAL 8K RAM CHIP PIGGYBACKED on top of EACH of the removed PIGGYBACKED PAIRS OF 8K RAM CHIPS connecting EACH PIN to its CORRESPONDING PIN below with the EXCEPTION of PIN 20 (CHIP SELECT)) BEND PIN 20 outward like PIN 20 on the CHIP below it. Reinstall these EIGHT PIGGYBACKED TRIOS into their sockets (U3-U6 and U12-Y13) and RECONNECT the ORIGINAL lines from PIN 20 of the CENTER CHIPS to their ORIGINAL POINTS on the EXPANSION JACK next to U3.
6. Install the ADDITIONAL 4 to 16 DECODER CHIP (74LS154) PIGGYBACKED on top of the ORIGINAL 4 to 16 DECODER CHIP, U2. Connect PIN 12 and PINS 20 THRU 24 to their corresponding PINS below. Bend PINS 1 thru 11 and PINS 13 THRU 19 OUTWARD. Reinstall the PIGGYBACKED PAIR OF 4 to 16 DECODERS in its U2 socket. Connect lines from the UPPER CHIP PINS 1 THRU 8 as follows.

PIN 1 to U1 SOCKET PIN 15. PIN 2 TO U1 SOCKET PIN 14.
PIN 3 TO U1 SOCKET PIN 13. PIN 4 TO U1 SOCKET PIN 12.
PIN 5 TO U1 SOCKET PIN 11. PIN 6 TO U1 SOCKET PIN 10.
PIN 7 TO U1 SOCKET PIN 9. PIN 8 TO U1 SOCKET PIN 7.

These provide the CHIP SELECT SIGNALS to the ORIGINAL (CENTER LAYER) of 8K RAM CHIPS.

Connect lines from the UPPER CHIP PINS 9 THRU 11 and 13 THRU 17 as follows:

PIN 9 TO PIN 20 U3 TOP 8K CHIP.
PIN 10 TO PIN 20 U4 TOP 8K CHIP.
PIN 11 TO PIN 20 U5 TOP 8K CHIP.
PIN 13 TO PIN 20 U6 TOP 8K CHIP.
PIN 14 TO PIN 20 U12 TOP 8K CHIP.
PIN 15 TO PIN 20 U13 TOP 8K CHIP.
PIN 16 TO PIN 20 U14 TOP 8K CHIP.
PIN 17 TO PIN 20 U15 TOP 8K CHIP.

These provide the CHIP SELECT SIGNALS to the ADDITIONAL EIGHT 8K RAM CHIPS (TOP LAYER)

7. Install a new NOR GATE (74LS02) PIGGYBACKED on top of the ORIGINAL NOR GATE, U10. Connect PINS 2, 7, and 14 to the CORRESPONDING PINS below. BEND PINS 1, 3 THRU 6, and 8 THRU 13 outward. Reinstall the PIGGYBACKED PAIR of NOR GATES in its U10 socket. Connect LINES from the UPPER CHIP as follows:

PIN 1 to PINS 18 and 19 U2 UPPER CHIP.
PIN 3 to U1 SOCKET PIN 6.

These provide the CHIP SELECT SIGNAL for U2 UPPER 4 to 16 DECODER CHIP thus fully decoding the available MEMORY ADDRESS LINES.

PINS 4 THRU 6 and PINS 8 THRU 13 of the UPPER NOR GATE U10 are not used and are left NOT connected. They can be used in future modifications.

This completes the HARDWARE modifications to the RAMDISK CARD. Next the DSR SOFTWARE must be modified so that this ADDITIONAL MEMORY can be accessed.

The original DSR CODE< CALL SUBPROGRAMS, ETC. are located in RACKS 90-92 at the top of the RAMDISK MEMORY MAP. The MODIFIED RAMDISK MEMORY MAP now extends to RACK 124 and the DSR must be moved to the new top, in RACKS 122-124.

NOTE: IF THE CODE IS NOT MOVED, IT WILL BE ERASED WHEN THE RAMDISK IS INITIALIZED TO MORE THAN 720 SECTORS.

The changes to the CODE consist of changing ALL REFERENCES for the 3 upper 2K blocks of memory to a NEW LOCATION, changing the LOADER PROGRAMS to LOAD the NEW CODE at the NEW LOCATION, changing the MEMTEST PROGRAM to check THIRTY-TWO 8K CHIPS, changing the MAX SECTOR CALL, and modifying the FORMAT ROUTINE of the DSR.

Luckily, this is much EASIER than it might appear since the SOURCE CODE for the HORIZON RAMDISK was provided with the KIT and is well Documented!

The following PROGRAMS will need to be modified and then REASSEMBLED with the EDITOR/ASSEMBLER. CALL/S, CHECK/S, CLEAR/S, CREATE/S, FILL/S, LOADER/S, PARTA, SVXB/S, XB/S, and VERSION/S.

The BASIC progras MEMTEST must also be modified. The other ORIGINAL SOURCE programs do not require modifications and are used as is.

1. MEMTEST - Delete LINES 110, 130 thru 170, 190, 200, 320 thru 340. Change LINE 180 from "LENGTH=24" TO "LENGTH=32".
2. CALL/S - Change "CI R2,1441" TO CI R2,977" AT LABEL MAI02.
3. CHECK/S - Change "CI R2,24" TO CI R2,32" (fourth LINE after LABEL CHK1).
4. CLEAR/S - Change "LI R2,90" to "LI R2,123" (fourth LINE after LABEL LOOP1).

5. CREATE/S -
Change "DATA >BB00" to "DATA >FB00" at LABEL LINK1.
Change "DATA >BD00" to "DATA >FD00" at LABEL LINK2.
Change "DATA >BF00" to "DATA >FF00" at LABEL LINK3.
Change "PARTA_03" to "PARTA246" in the TEXT LINE after LABEL PDATA.
Change "PARTB_02 to "PART B256" in the TEXT LINE after LABEL LDATA.

6. FILL/S - Change "LI R5,93" to "LI R5,125" (one LINE before LABEL FLOOP1).

7. LOADER/S -
Change "DATA >BB00" to "DATA >FB00" at LABEL LINK.
Change "BYTE >BB" to "BYTE >FB" at LABEL MXL1.
Change "BYTE >BD" to BYTE ">FB" at LABEL MXL2.
Change "BYTE >BF" to "BYTE >FF" at LABEL MXL3.

8. PARTA -
Change "DATA 720" to "DATA 976" at LABEL MAXSEC.
Change "DATA 720" to "DATA 976" at LABEL FORSEC.
Change "DATA >BB00" to "DATA >FB00" at LABEL LINK1.
Change "DATA >BD00" to "DATA >FD00" at LABEL LINK2.
Change "DATA >BF00" to "DATA >FF00" at LABEL LINK3.
Add the LINES "C R8@MAXSEC" and "JEQ FFDONE" after the LINE "INC R8" (fourth LINE after LABEL FMTLP1.
Add the LINE "FFDONE MOV 48,R3" after the LINE "JNE FMTLP0" (sixth LINE after LABEL FMTLP1.

9. SVXB/S - Change "LI R1,>BF00" to "LI R1,>FF00" (fourth LINE after LABEL SVXB.

10. VERSION/S -
Change "DATA >BB00" to "DATA >FB00" at LABEL LINK1.
Change "DATA >BD00" to "DATA >FD00" at LABEL LINK2.
Change "DATA >BF00" to "DATA >FF00" at LABEL LINK3.
Change "PARTA_03" to "PARTA256" in the TEXT LINE after LABEL PDATA.
Change "PARTB_03" to "PARTB256" in the TEXT LINE after LABEL LDATA.

11. XB/S - Change "CI R2,1441" to "CI R2,977" at LABEL MAX02.

REASSEMBLE these FILES to create the NEW OBJECT FILES. Then REASSEMBLE the FILE "TEST/S" which ASSEMBLES the DSR FILES PARTA thru PARTE. Call this FILE "DSR256". ASSEMBLE the ORIGINAL FILES "CHAR/S" and "DOWNLD/S" from the HORIZON SOURCE DISK.

Next RUN the "LOADER" program assembled from "LOADER/S" to LOAD the following:

"DSR256" into BLOCK 1.
"CALL" from the assembled FILE "CALL/S" into BLOCK 2.
"CHAR" from the assembled FILE "CHAR/S" into BLOCK 3.
"DOWNLD" from the assembled FILE "DOWNLD/S" into BLOCK 3.

Now run this BASIC PROGRAM:

```
100 CALL INIT
110 CALL LOAD("DSK1.XB")
120 CALL LOAD("DSK1.SVXB")
130 CALL LINK("SVXB")
140 END
```

NOTE: the RAMDISK MUST be set at CRU 1000 for the SVXB program to work as it does NOT search for the HORIZON CARD CRU like the other programs do. If you have another CARD at CRU 1000 (like the MYARC 128K or 512K CARD), you can change the sixth LINE of the "SVXB/S" FILE from "LI R12,1000" to "LI R12,(CRU of your HORIZON CARD)".

The modified DSR CODE, CALL SUBPROGRAMS ETC. are now LOADED in their NEW locations in RACKS 122 THRU 124. Next RUN the program "CREATE" from the assembled FILE "CREATE/S" (PROGRAM NAME "IMAGE")) This will create the FILES "PARTA246" and "PARTB256" on DSK1 for use with the VER 256 LOADER from the assembled file "VERSION/S".

The SOURCE CODE for the program "UTIL1" (the multiple RAMDISK LOADER) was not provided on the HORIZON SOURCE CODE DISK. It can be modified by DISSASSEMBLING it with "MILLERS GRAPHICS DISkASSEMBLER", finding the four words >02D0, >BB00, >BD00, and >BF00, changing them to> >03D0, >FB00, >FD00, >FF00, changing "PARTA_03" and "PARTB_03" to "PARTA256" and "PARTB256", REASSEMBLING the program, and then RUNning the SAVE UTILITY from the EDITOR/ASSEMBLER to change it back to PROGRAM IMAGE FORMAT.

This completes the DSR modifications. All functions of the HORIZON RAMDISK will function as they did originally, but now being able to UTILIZE 976 Sectors (256K).

When formatting the 976 SECTOR RAMDISK, select DSDD format. The DISKMANAGER will show "974 SECTORS FREE" and "466 SECTORS USED". This is because the DISKMANAGER is trying to format 1440 SECTORS and reads 466 USED during SECTOR VERIFICATION. This does not affect RAMDISK OPERATION in any way, but it can be corrected to show "974 SECTORS FREE " and "2 SECTORS USED" by changing BYTES 10 and 11 of SECTOR 0 from >05A0 to >03D0. The following program is used to correct the SECTORS FORMATTED number.

```
        DEF START
SECTOR  DATA >03D0
START   LI   R12,>1200 CRU OF YOUR CARD
        LI   R1,7
        SWPB R1
        LDCR R1,8
        MOV  SECTOR, @>580A
        SBZ  0
        RT
        END  START
```

This completes the HORIZON RAMDISK 256K EXPANSION PROJECT. Questions concerning this project should be sent to: EDWARD A. HALLETT, 5600 S. COUNTRYCLUB #64, TUCSON, AZ 85706. Phone (602)889-6930.

BY ART BYERS, WESTCHESTER, NY

The fourth annual TI Faire hosted by the Chicago TI UG was a fine success from all points of view: Around 900 99/4A owner/users from the middle of the nation (and as far away as Boston, Washington, DC and New York) had a chance to see the newest and best in software and hardware and to spend some money on goodies. So- the Vendors obviously did a great deal of business and will be encouraged to attend other TI shows around the nation. The speakers and events were interesting and well run. Those attending learned must that was new.

Best of all, however, was the chance to meet and mix with the 99'er community, to exchange ideas and information, and to renew our enthusiasm as we proved once again the motto originated by Henry Heim: "We may be an Orphan, but we have a great future!".

For me, the occasion meant putting faces together with many of the famous names such as Chris Bobbit (Asgard), Jim Horn and Jeff Guide (Disk Only Software, Ron Albright (The Orphan Chronicles), Theresa Masters (president of the LA UG), -plus well-known programmers such as Peter Hoddie, Coe Case, Todd Kaplan, and Paul Charlton, - oh yes! and the Tigercub Himself and the most well-known of them all, Jim Peterson. Without Jim half of most UG newsletters would be blank space! It was nice to talk to Walt Howe and Bob Demeter in person, instead via Telecom.

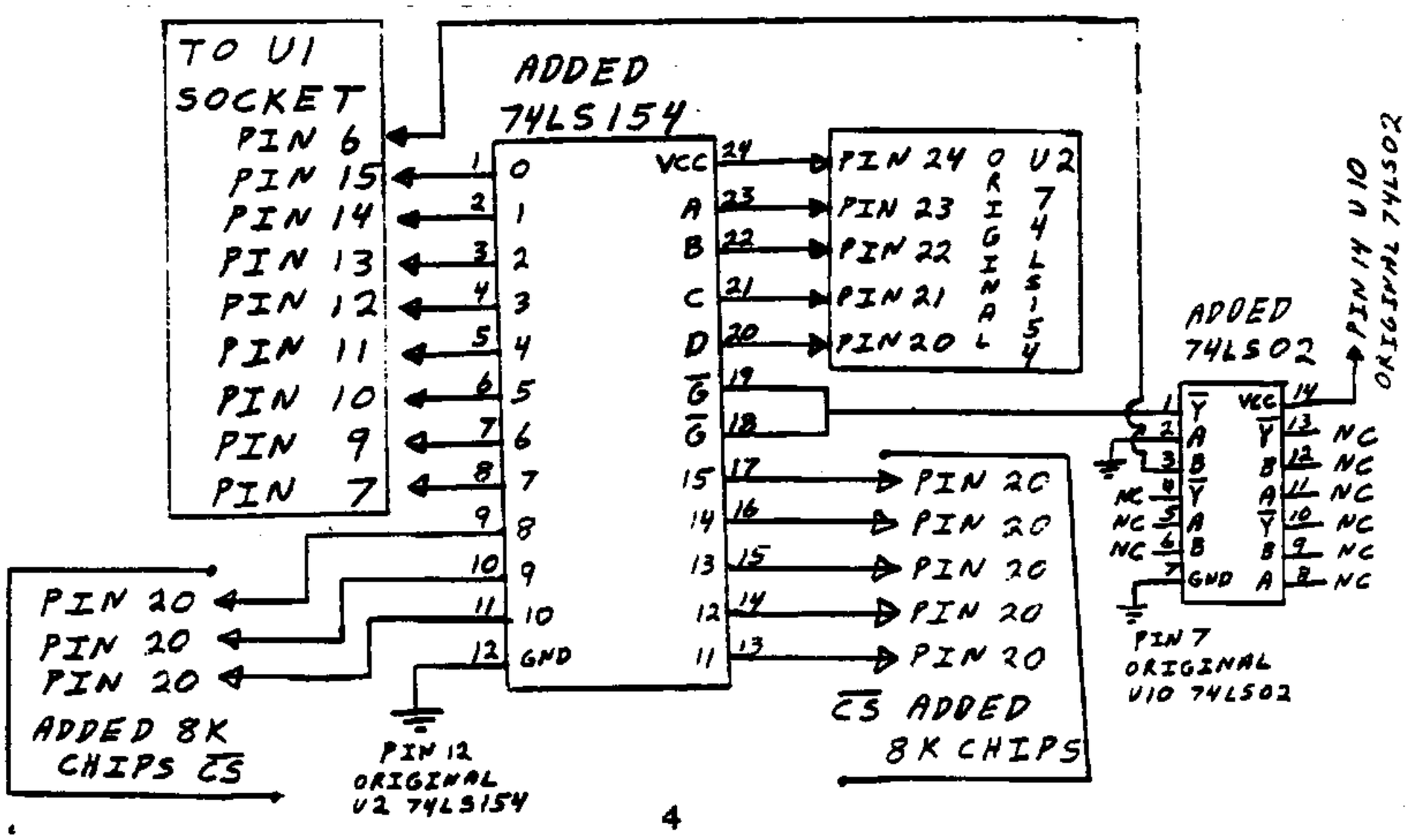
What was new? Oh Boy! Lots was new! J. Peter Hoddie has a new graphics program "Font Writer" being marketed by Asgard. ("What! another graphics program" you are about to say) - but you have to see it to believe it. That was only a start. There must have been 50 new software items offered by vendors covering everything from games and education to disk utilities and home finance. DataBiotics long-awaited PILOT language with extensive documentation was finished and on sale plus Super Forth. New Hardware?? Yes and plenty of it! Mini PE Boxes for 3,4 or 5 cards, and prototypes of everything from a sort of super widget that enables you to use the "Review Module Library" hidden software built into the console to a whole new computer (Myarc 9604). One of the big hits of the show was the new IBM type keyboard for our 99/4A, made by RAVE Mgf. I have an interview on tape with one of the principals involved and will print it in detail in the CALL SOUNDS newsletter.

During the lunch break, there was a fascinating rap session for the visiting UG representatives. Everything was covered from rumors of more software to be released to PD and UG's by Texas Instruments (a diagnostic disk) to fundraising and dues policies, methods of software library distribution, and newsletter exchange.

One of the best talk/demonstrations of the day was the hour on music and the 99/4A, put on by J. Peter Hoddie. That included two linked computers playing music together, music that played while you are programming, and two pieces where Peter played the Cello accompanied by the 99/4A!!!

All in all the Chicago club is to be highly complimented. Everything ran smoothly. Those of you who could not make this show are advised to make the next one near you, be it Seattle, Boston, Dallas or New Jersey. These fairs are FUN!! I will report in detail at the November club meeting.

For Ramdisk modification



ODDS and ENDS

By Bill Gaskill, Front Ranger, October, 1986

Some of what you will read here, you may have seen before, some perhaps you haven't seen. All of it is simply a collection of ideas, information and tips about our computer, accumulated over the last three years. It is not presented in any particular order, being written down as it came to mind. I hope you enjoy it.

1. Ti's Catalog program on Programming Aids I will write a DV/80 file that can be read by TI-Writer. If you want to create a list of the contents of all your disks, that you can look at later and find anything in the list that you want, you can use this Catalog program to do it. Simply choose option 4 from the catalog's menu and send the output to DSK1.CAT and each disk will catalog itself and then write a file to itself (named CAT) that can then be loaded into TIW for later reference. Each CAT program must be merged into the existing file by using the LF function and then specifying a line number after which the new file is to be positioned. For example, if you have the first disk's CAT loaded into TWI and it ends at line 15, you would merge the next disk's CAT file into TIW with the following keystrokes: FCTN 9 - to activate the command mode; LF and then <Enter> - to Load File. 1 E 16 DSK!.CAT - to merge the file named CAT from the disk in drive one, encoepassing lines 1 to the end of that file, placing it after line 15 of the CAT file already loaded. <ENTER> to activate the merge. Finding which disk contains which program or data file is simply a matter of using the ReplaceString option from TIW's command mode.

2. A program written in Extended Basic, where you have created line numbers out of sequence, can be made to run faster. You must first SAVE the program in MERGE format, then type in NEW and the reload the program using MERGE. The MERGE routine cleans up the program by picking up each line in its proper order, where as your original program's line number table was saved in the order that you created each line in, regardless of the number of the line.

3. REDO can be used to repeat any command typed in at the operating system level (the > prompt) such as SAVE, OLD, PRINT, LIST, CALL INIT, CALL LOAD, DELETE, etc.

4. RUN can be used with CS! to redefine a character set with one program and then RUN another program without "killing" the new character set or robbing memory from the second program being loaded.

5. 10 M$="THIS IS MY PROGRAM"
   20 FOR I=28 TO 6 STEP -1 : : DISPLAY AT(2,I):
   SEG$(M$,1,X) : : X=X+1 :: NEXT I
will make the string m$ appear on the screen from the upper right, in row 2, one character at a time, until it stops in Column 6.

6. Danny Michael's NEATLIST program can be used to print out a listing of any XBASIC program regardless of protection status or line number table alteration, etc.

7. You can alter the color of your cursor to white on blue (or any other color combination) by including a CALL COLOR(0,16,5) statement in your program.

8. A subfile can be selected from a memory-resident data file, and saved to disk, by creating an array for the record numbers (array pointers) of the selected records and then saving those records referenced by the array pointers.

9. John Hamilton of the Central Iowa Users Group offers a 22-page booklet of TI Tips for only $4.00: John Hamilton, CIUG, Box 3043, Des Moines, IA 50316.

10. Bob Lawson offers a FREEWARE utility that will print out a Household Budget Management file (This is in the Cleveland Groups' libraries, ED)

11. LIST "DSK1.PROGRAM", where PROGRAM is the name of your program will create a DV/80 listing of your Xbasic program that can be read by TI-Writer.

12. Pressing FCTN V when SAV(ing) an XB program will save the program with an invisible name. The underline character is actually what is written to disk (ASCII 95 or >7F), although the delete character (ASCII 127) is what is read. Up to 10 files could be masked on a single disk in this way.

13. Pressing FCTN X 10 times from any menu screen in Disk Manager II will invoke a proprietary protection routine that will be written on the disk you initialize so that it cannot be copied by another DMII module.

14. Both TRACK-HACK and TURBO2 can be loaded onto your Funnelwriter disk and be RUN from the Utilities option. Pres 3 from the main menu. Press 9 and then use option 3 for TRACK-HACK and option 4 for TURBO2. It saves swapping to he E/A module (unless you have GramKracker).

15. 4A/TALK can upload DV/80 files to CompuServe's TI FORUM in 7-bit ASCII. The results are screen readable but NOT downloadable in readable format.

16. Terminal Emulator III (That's right, 3!) has a very nice CHAR1 file for true lower case letters that is much easier to read and much better looking than the true lower-case upgrade that TI put our for TI-Writer.

17. The NEC JC-1225MA color monitor can be used with a TI, even though it is not specified in the instructions. The Red wire plugs into the "Video In", the yellow wire to "Audio Out."

18. A protected XB program on cassette tape can be duplicated using two cassette records set to maximum volume.

19. Records/files that have been DELETED from a disk are not actually deleted. Only reference to their existence is deleted, not the actual information.

20. J. Peter Hoddie has rewritten DM1000 and Fast-Term so that they (either one, not both) can be loaded into GRAMS 1 and 2 in Gram Kracker. The programs are available on Barry Traver's Genial Traveler 1.3.

21. You can make a message flash or flutter on screen any time you use a CALL KEY statement by including a DISPLAY AT statement in a loop created by the CALL KEY. An example is shown below.

10 CALL KEY(0,K,S) :: DISPLAY AT(12,2):"THIS IS THE MESSAGE" :: DISPLAY AT(12,2):" " ::IF S=0 THEN 10

This is only a short list of the no doubt thousands of "things" TI users have discovered on their own. Maybe you can add to the list?

BASIC PROGRAMMING
OZARK 99'ER NEWS VIA WEST PENN 99ER
OCTOBER, 1986

This month's Basic column is devoted to short routines to help you display things on the screen where you want them. (These routines originally appeared in an article by Marshall Gordon in the A9CUG Call Newsletter.)

1) to center a line of text, use the following command:

```
PRINT TAB(27-LEN(A$))/2;A$
```

This will print any string called a$ (up to 28 characters) centered on the screen. No more need to count and tab each line individually. Here is an example:

```
 10 DATA OZARK, 994/A COMPUTER, USERS GROUP, STOP
 20 CALL CLEAR
 30 GOSUB 5010
 40 GOTO 40
5000 REM $CENTER LINES OF TEXT$
5010 REM
5020 READ A$
5030 IF A$="STOP" THEN 5100
5040 PRINT TAB(27-LEN(A$))/2;A$
5050 GOTO 5020
5100 RETURN
```

2) You can also TAB vertically by adding the following lines to the program above:

```
5010 CU=0
5030 IF A$="STOP" THEN 5070
5040 CU=CU+1
5070 FOR I=1 TO (24-CU)/2
5080 PRINT
5090 NEXT I
```

This can be used as a subprogram with the data strings anywhere in the program. The centering routine can be put at the end of your program and accessed by a "GOSUB" command. Then when you get to a place in your program that you want text to be printed and centered on the screen, all you have to do is write the strings as "DATA" statements with the last word as "STOP" and a "GOSUB" to 5100 (or whatever line your subroutine begins at) and you are all set. One final word...if you want a "spacer" line between printed lines, try this:

```
 10 DATA OZARK,"",99/4A COMPUTER,"", USERS GROUP, STOP
```

3) Next we'll see how to write a line anywhere on the screen (like the DISPLAY AT command in XBASIC). First, a word of caution. This procedure uses a 32-column screen. Those of you using TV's as monitors may not see columns 1,2,31, and 32 too well. All 24 vertical positions should be easily seen, however. Try the following program:

```
 10 CALL CLEAR
 20 A$="OZARK 99'ERS"
 30 ROW=10
 40 COL=8
 50 GOSUB 5010
 60 GOTO 60
5000 REM $PRINT ON SCREEN$
5010 REM
5030 FOR I=1 TO LEN(A$)
5040 CALL HCHAR(ROW,COL+I,ASC(SEG$(A$,I,1)))
5050 NEXT I
5060 RETURN
```

Here's how the routine works. Line 5030 begins a FOR NEXT loop from 1 to the 'len'gth of a$ (..LEN(A$)=the number of characters in a$). Line 5040 uses the ASC command to convert the characters in a$ into their ASCII codes one at a time, and then print each one out on the screen beginning at the specified position. All you need to do is define ROW, COL, and A$ and do a GOSUB to the print routine. By the way, when you define A$, use a space as the last character in the string. It makes the string print smoother and faster (but no one seems to know why)>

Now, those of you who are a little lazy (as all good programmers seem to be), may have noticed that it takes 3 lines to set up each line to be printed..one for a$, one for ROW, and one for COL. There should be a way to do it all in one line, shouldn't there? Well, there is. What we need to do is first "pack" all of our information into a single string. For example:

```
 20 A$="1008OZARK 99'ERS " (Delete lines 30 and 40)
```

But now that we've packed it, we need to be able to "unpack" it, too. The following lines will accomplish that for us. Add them to the program above.

```
5010 ROW=VAL(SEG$(A$,1,2))
5020 COL=VAL(SEG$(A$,3,2))
5030 FOR I=1 TO LEN(A$)-4
5040 CALL HCHAR(ROW,COL+I,ASC(SEG$(A$,4+I,1)))
```

Line 5010 takes the 'SEG'ment of A$ beginning with the first character and counting over two characters--in this case SEG$(a$,1,2)="10". Then we 'VAL'ue the result and wind up with the number 10. (Remember that the string "10" cannot be used as a number. You must convert it with the VAL command first.) Then ROW=10.

Line 5020 does the same for the two characters of a$ beginning with the third character (08), so COL=8, the first four characters are not to be printed, the number of characters to print equals LEN(A$)-4.

Line 5040 is the same as before except we begin printing at character 4+i of the string. (Remember the first time through the loop i=1, so we begin printing at character 4+1 or 5.

PRE-SCAN IT!
A REVIEW BY DUANE GOODMAN
WORDPLAY, 11/86 (PORTLAND UG)

This fine program is written by J. Peter Hoddie and is distributed by Asgard Software.

The program takes an existing Ex-Basic program, re-writes it in pre-scan format, and inserts the appropriate coding to turn on and off the Pre-Scan feature when necessary. It also will replace up to 5 numbers in the program with the variables "", "[", "]", "_", and "\". It takes only 1/3 the space to store this type of variable as opposed to the number itself. Pre-Scan IT will also remove all the "REM" comments and replace them with exclamation points to save program space.

Using Pre-Scan IT!

The first thing you do is RESequence the program you wish to Pre-Scan. Pre-Scan IT needs the area from 1 to 100 for its own use. After Resequencing the program, save it in the following format:

SAVE DSKn.XXXXXXXXXX,MERGE

Where XXXXXXXXXX represents a filename different than the original program. After saving in MERGE format, insert the PSI disk and type the following to load and execute Pre-Scan IT:

"RUN DSK1.LOAD"

After running the program thru Pre-Scan IT, it will again be written to the disk in MERGE format. To return it to PROGRAM format, you must type:

OLD DSKn.XXXXXXXXXX

After it is loaded, you would type:

SAVE DKSn.XXXXXXXXXX

and your file will be written in program format ready to use.

I used Pre-Scan IT on an AUTO MAINTENANCE program I had purchased which was slow. The initial loading time for the program was 30 seconds. After running it thru Pre-Scan IT, it loads in 18 seconds. That is a 40% increase. There is also the same speed increase going from one menu selection to another within the program.

If you either use a lot of Ex-Basic programs or programs in Ex-Basic, this program is well worth the $10.00 asking price.

To order send $10.00 to:

ASGARD Software
P.O. Box 10306
Rockville, MD 20850

For more information or to order by credit card, call 301/345-2492

CLEVELAND AREA 99/4A USERS GROUPS
C/O DEANNA SHERIDAN
20311 LAKE ROAD
ROCKY RIVER, OH 44116

PLEASE NOTE - NEW ADDRESS

CHECK YOUR EXPIRATION DATE.
THIS MAY BE YOUR LAST ISSUE!

!! TIME DATED MATERIAL !!