CLEVELAND AREA 99-4A USERS GROUPS NEWSLETTER

JULY/AUGUST 1987

OFFICERS	NORTHCOAST	TI-CHIPS	MEETING DATES	
PRESIDENT	MARTIN SMOLEY 1-257-1661	TERRY VACHA 225-5368	NORTHCOAST 1:30 P.M.	TI-CHIPS 10:00 A.M.
VICE PRESIDENT	RICH JOHNSON 261-9274	RUSS SHIMANDLE 1-887-5330	EUCLIDIAN ROOM	NORTH ROYALTON LIBRARY
TREASURER	JIM MEKEEL 286-3179	LIN SHAW 235-3912	EUCLID SQUARE MALL	STATE ROAD & RT 82
MEMBERSHIP	ELMO IACOBUCCI 585-2588	JOHN PARKEN 331-2830	THIRD SATURDAY	THIRD SATURDAY
	2161 Pine Ridge Drive	4172 W.217TH ST.		
	Wickliffe, OH 44092	Fairview Park, OH 44126	JULY 18, 1987	JULY 18, 1987
SECRETARY	CHUCK POULIN 731-6473	MARY PHILLIPS 592-4009	AUGUST 15, 1987	AUGUST 15, 1987
LIBRARY(DISK)	ERNIE & DON NITSCHKE 888-4845	MARK McCAULEY 235-8888	SEPTEMBER 19, 1987	SEPTEMBER 19, 1987
(TAPE)	TOM NELLIS 475-4067	JOHN PARKEN 331-2830	OCTOBER 17, 1987	OCTOBER 17, 1987
	BBS 216-944-1072 (24 HRS)		NOVEMBER 21, 1987	NOVEMBER 21, 1987

Remember, there will be NO newsletter in August, Thank goodness! Hopefully this issue has a variety of programming articles to do until September.

If you have not filled out the survey by Ali Ulgen, please take a few minutes to do so and send it off to him. He hopes to have a cross section of about 1,000 TI users on which to base his analysis and at the moment has only about 400. He is greatly disappointed in that some of the larger groups have refused to distribute this, saying they do their own survey, or else do not use outside materials. These comments are coming from groups who expect US to support THEM when they have fairs, etc. I guess he has been accused of all kinds of sinister actions on The Source, when he is just an avid TI user who hoped to put together a picture of what is going on in the TI World. Let's at least give him 100% support on a local basis.

Jim Swedlow (Sideprint) made the comment in his column in the ROM this month, he feels the TI has about 2 more good years life left. We are an endangered species! Perhaps some clubs feel they have even less time than that. Many have been existence about 5 years now and unfortunately, in many of those clubs, the same people have been carrying the load. They are burned out, tired and to the point that if no one else cares and is willing to step in, those clubs are on the brink of extinction today. That is why it is so very important we have a LOT of people doing a LOT of little things, instead of a FEW people doing EVERYTHING. Think about what you can contribute. We have another sub-librarian in that Paul Newmeyer is going to tackle anything he can find about FORTH and Harry Hoffman is doing FAIRWARE.

At the Northcoast group several people said they would donate modules for the project noted in last month's newsletter. We have a potential of about 50 with the few people we talked with. Now, we need a person to take care of them and check them out. How about it?

More on ASCII files. Last month we featured programs putting other programs into ASCII files. You can also do a lot of tricks by creating files in ASCII and printing them out with XBASIC programs. That is how I do the newsletter in double-column format. I found a neat calendar program in one the ROM in which you create you calendar in TI-Writer and print it out with a Basic program. It would not be a notepad type calendar, but rather the type you would want to print for a month's activities for your school, or other type of organization. I have typed it in and will get it to the library in due time.

Tip: If your TI-Writer program defaults to DSKX. at the top of the screen, you can use FCTN 7 and it will tab over to where you want to type in the program name, and you won't have to space over one letter at a time. This is courtesy of Wes Richardson of Lexingnton, KY. (You always learn at least one neat trick with a group of TI people)

CONTENTS

TI CHIPS EXECUTIVE NOTES
JAN FEDOR

There was a nice turnout of members for our June meeting despite the rain. We have been getting some new members and visitors each meeting thanks partly to the postcards being sent to TI owners inviting them to attend our meetings. Our thanks to Russ Schimandle and Harry Hoffman.

Tom and Judy Thalner brought the MYARC 9640 and tom demonstrated and explained all the DOS commands and showed how to load TI programs. It does not use cartridges which must first be converted to disk. There are a number of software people working on programs for this computer, such as TI Artrist, Printer's Apprentice and others.

John Parken disassembled his computer (as did Vonn Malcuit) to show how to put 32K chips in. John used 4 8K chips and Vonn had 1 32K chip. John has written instructions for those interested. Glenn Bernasek stated if anyone was interested,to contact him. Possibly one order could be sent in. Glenn's number is 238-6335 after 6 p.m.

Les Kee gave a demonstration of a Basic program, SUBPROGEX, Subprogramming in Extended Basic. It dealt with handling variables from the main program to sub-programs.

Matt Andel showed Artist's Compansion and how to interface Supersketch with TI Artist.

Terry Vacha showed a program for drawing with Joysticks. He also demonstrated SST Compiler which allows you to type a program in Basic, then the SST turns it into a code that the Editor Assembler recognizes. Very handy.

A raffle was held for 15 programmed disks. The winner was a new member, Frank Bardy. Congratulations, Frank.

The next meeting will be July 18.

HOW TO LOAD SUPER SKETCH INTO TI-ARTIST
MATT ANDEL - TI-CHIPS

1. Plug Editor/Assembler, Extended Basic, or Mini Memory in the middle slot of the cartridge expander.
2. Plug the Super Sketch cartridge in one of the other slots.
3. Select Editor/Assembler, Extended Basic, or Mini Memory.
4. Load TI-Artist the same way you always do.
5. When you get the main menu, press 3 for input device.
6. Put the Artist Extras Disk in drive 1, 2, or 3.
7. Type DSKx. SKETCH, then PRESS ENTER.
8. When done loading, most the switch on the cartridge expander to the Super Sketch Cartridge.
9. If you use Drive 1 to load Sketch, but the Artist disk back in Drive 1. You won't need the Extras disk any more.
10. Press 1 for TI-Artist and you can now use super sketch to draw with instead of your joystick or keyboard.
11. On Super Sketch you use the select button to select the function you want.
12. On Super Sketch you use the select button to select the function you want.
13. On Super Sketch you use the lift button to draw and to activate the function.
14. If you want to return to the main menu, press FCTN QUIT. Now, you can use number 2 enhancement, or go back to 1 Ti-Artist.

EXECUTIVE NOTES - NORTHCOAST

If you did not attend the last meeting you missed a pretty good one. The demo by Frank Jenkins was very informative. Multiplan is difficult to get into if you have never used before, but Frank made it quite understandable. The people who attended must have come specifically for Multiplan, because they watched intently, and afterwords they asked some very good questions. I think that some of them already knew a great deal about Multiplan I found this part of the meeting very enlightening.

I would like to take a moment to thank all of the people who help with the meetings, wether they bring and set up equipment, or set up tables and chairs at the hall.

Writing this article for the newsletter reminds me that we should all thank Deanna as often as possible for the massive amount of work she puts into getting this newsletter out. Without her endless effort I am sure that we would not have a newsletter. ***>> Thanks Deanna. <<***

This brings me to the subject of

VOLUNTEERS

We are looking for volunteers for just about every group activity you can imagine. The first item is sub-librarians. We need people to help with the library on a permanent basis. The sub-librarians would pick a specific subject in which they are interested, such as c99, E/A, Music, etc. All disks we receive in those areas would be given to that person or those people in that area. Those people would analyze and catalog the disk and then return it to the main library. This would allow you to get first crack at new software in an area where you have special interest, and also take some of the pressure off the people who presently do this job.

Paul Newmeyer has already volunteered to take care of the Forth sub-library for the club and continue to write those great Forth articles for the newsletter. "That's great."

IMPORTANT

Concerning The BBS

The executives of NorthCoast are considering pulling the plug on the bullitin board. This is due to a severe lack of use by the membership and a multitude of other problems, plus the expense of the phone bill etc. If you have any reasons why we should keep the board, or any suggestions, **WRITE** to me personally.

Note: Don't make suggestion that we should do this or that unless you are willing to help on a personal basis. It's easy to say that we should do things and then expect someone else to do it. We need more (Suggester-Do'ers) in this group.

The Next NorthCoast Meeting

The next meeting will feature a demonstration by Pa Wheeler of the new program " PC Keys II ". This software is quite unique as to the customization of your keyboard input. And the material is not as heavy as we have had recently, so it should make for a light and interesting meeting.

See you all at the next meeting. Marty.

EXCERPTS FROM J.PETER HODDIE
REPORT TO BOSTON COMPUTER SOCIETY
JUNE 17, 1987

As of late I have been in what some people call "hiding." I spent last week in New York State with Paul Charlton, working 12 to 18 hours a day to finish the operating system for the 9640 (I am mostly writing device drivers for the various peripherals, while Paul is doing the operating system itself). There were a couple of sleepless nights spent staring bleary eyed into monitors, lots of consumption of liquid caffeine , several phone calls from people wanting to know when DOS would be finished (including MYARC's secretary Cynthia, who wants to know what to tell the flood of callers to MYARC looking for the OS), and lots of bugs created and later eliminated . The bottom line is that the OS is still not done. It is one huge project. We had hoped to have it ready to show at this month's BCS meeting , but it didn't work out. Certain parts of the system we had hoped to have finished last Wednesday, weren't really close until Sunday morning. Work goes on. Slowly. Tediously. But it continues. Believe me, no one wants this operating system to be finished more than Paul and myself. The work is slowly driving us crazy. The operating system could be finished in 24 hours, or it could take rather longer. We ain't stalling. There is just huge amounts of code to integrate. The operating system is about 88K in size. That is larger by nearly a factor of 3 than the largest program you can create on the standard 4/A system.

On a related note, I would like to set the story straight on hardware compatibility with the 9640. First of all, the TI, CorComp, and MYARC disk controllers will all work. It doesn't matter which EPROM you have in the card. The TI controller can handle 80-track drives (just not in double density mode), and the CorComp controller and the MYARC controller can handle 80 track, and 16 or 18 sectors per track. The reason for this is that the EPROM or ROM in the disk controller is not used by the 9640, but is replaced with code included in the operating system. This allows the TI and CorComp controllers to run as fast as the MYARC currently does. The speed of disk access is really impressive - you may not recognize your disk drives. Any RS232 card from TI, MYARC, or CorComp will work. Print spooling is built into the system for all cards, and the size of the spooler can be set by the user. The print spooler is accessed just like a normal device, such as PIO, rather than SPPIO as on the MYARC 512K card. The Horizon RAM disk will work; however, at this time in order to boot the system from it, it must use the Horizon EPROM from Genial Computerware. This is not a ploy for me to make lots of money, but a decision made because of several unfortunate characteristics of the ROS distributed with the Horizon card. Currently there is support for only 1 Horizon RAM disk, although this could change in the future. The MYARC 512K card cannot be used as is. However, for $15 MYARC will convert it so it can be used as additional memory for the 9640. Once this change is made, the 512K card cannot be used with a /4A, so carefully consider having this modification made. The speech synthesizer is supported but you have to buy a special card to put into the expansion box. Such a card is available for about $40 from RAVE99. Your TI 32K or other memory expansion cards such as the Foundation will not work. Since the 9640 has over 600K of memory in its minimal configuration, this should not prove any great hardship. At this time, the Mechatronics GRAM card is not supported. The CorComp triple tech card will work, except that because of a somewhat faulty hardware decision (works on the /4A but not the 9640) the triple tech card will eat up about 1/8th of your available memory. The 9640 also supports an internal RAM disk which can be set to any size by the user, within the constraints of available memory. The current MYARC winchester personality card is supported, and, of course, the new MYARC hard drive/floppy controller will be supported when it becomes available. I hope this paragraph has cleared up any misunderstandings you may have had about the 9640 and your present hardware set up. Please let me know if you have any further questions.

The documentation on the 9640 doesn't currently mention some of the more interesting features that are in the computer. For example, all disk files are time and date stamped at creation and on every update. This information is available on disk catalogs, ad even from BASIC using an extension of the current method of cataloging a disk. The RAM disk support is done similarly to the MYARC MPES (mini peripheral expansion system), in that if you assign the internal RAM disk to be drive 1, you can then make your physical drive 1 respond as drive 2. This means that all drives can be made always available, which is not possible on the /4A. This is done independently of CRU base, thanks for the single master DSR (device service routine) created for the 9640. For the assembly programmer, there is a wealth f system utilities for graphics available through XOPs, written by Chris Faherety. The operating system also supports a new powerful set of disk access commands designed by Paul Charlton, and implemented by both of us. These allow for easy file and disk access from from assembly for disk and file copying and comparing. The operating system also supports multi-tasking when not in /4A mode. This means you could be editing a file with your word processor, while downloading a file from a bulletin board, while a graphic image of a frog dances on the corner of your screen. Multi-tasking allows you to run several programs at once - and this should open up some exciting possibilities in the future.

Until the operating system is released for the 9640, I would recommend taking anything you read from outside MYARC sources with a large grain of salt. That is to say, without naming names, that I have read numerous articles on the 9640 which contain information that is just plain wrong. The articles claim that the machine can't do certain things, or that it will eventually do some things better than it does now - and they are just completely wrong. While articles on the 9640 by people who have them at this stage are rather popular because people are crying out for any information they can get, many of those writing are very badly informed. This problem is as much a fault of MYARC as anyone . To release the hardware with incomplete software to anyone but developers was a serious mistake in my estimation. It has

calmed many people down, but it has started a new furor over "where is the operating system" which is just as bad as the old "when will it be released." Lou Phillips has a habit of saying things to calm people down. If someone asks him when a product will be ready, he tends to give the absolute best case answer. Unfortunately in this business, that tends to be way off base.

On to other subjects, I have not been devoting all my efforts on MYARC projects. As some of you may know, I wrote a program called Font Writer, and I promised an update to it this spring. Well, because of all the MYARC work, the project fell behind. However, beta testing of the update has only turned up one bug (barry Traver found it...one of his talents) which was quickly squashed. Front Writer II should be available in about 10 days. It now includes an improved manager, a significantly enhanced formatter that allows for boxes, proportional spacing, right justification, use of TI-Artist and GRAPHX pictures, and much more. There is also a pretty slick banner program, a dsk dump program to print a convenient reference sheet of all graphic files on a disk, and TI-Writer loaders. The program also includes much more assembly language for speed. It retails for $25 from Asgard. If you have the current version, an update is $6 for a new disk and manual. DataBioTics is currently working on a "super ram disk" which will feature something like 500K of memory that can be used as a RAM disk or several smaller RAM disks, and a print spooler. The product will also have a clock option to time and date stamp files, and should be 9640 compatible. If they can pull this one off, it looks to be a winner.

I guess this one qualifies as a "product announcement." John Johnson, creator of the popular MENU program for the Horizon RAM Disk, is preparing to release his first commercial program through Genial Computerware. The product, tentatively called "Remind Me!" is a calendar type program written 100% in assembly. It keeps one full month in memory at a time, and you can enter up to 12 forty column lines for each day using a TI-Writer style editor. This is a scratch pad where you can keep notes, and a facility is provided for moving information from one day to another in the same month or a different month. You can print out a report for the entire month or just selected days. You can search the month for multiple keywords, and just print days with those words. The program is incredibly fast , visually attractive, and very user friendly. If you have a CorComp clock card, MBP clock, Clulow clock, or a 9640, the program will automatically display the time and date on the screen. "Remind Me!" should be available in the next month for about $15.

THAT OBSOLETE COMPUTER EQUIPMENT MIGHT JUST GROW YOU SOME GREENS

Readers, wake up! I'm going to make you rich. No, I'm not going to tell you to invest in small, up-and-coming software start-up. Naah. That's for rubes. I've got a much better suggestion.

As most of us know, the stock market any day now, is set to take a plunge that's going to make 1929 look like a Comdex bash. And before it's over, we'll all probably be out on street corners selling Apples - or, worse, Commodore Amigas.

One way to put a little something in your nest before that stormy day hits is to make a wise investment in what financial experts call "tangible assets." This investment category includes real estate, stamps, coins, classic cars - anything that's likely to rise in value as good old - fashion American greenbacks become more worthless than glo-in-the-dark "CP/M Forever" bumper sticker.

So here's what you should do: Invest in old, obsolete computers. That's right, those digital boat anchors that are cluttering closets, basements and garages nationwide will, someday, be worth alot of money. You Laugh? You Scoff? Well, consider that the Duesenbergs that now sell for millions of dollars fetched only about 500 bucks as recently as 25 years ago. A mere 10 years ago, you could buy a Duesie for less than $50,000.

As with any investment, to make an eventual killing in collectable computers, you have to be a knowledgeable buyer. It makes no sense to purchase a machine that has no history behind it, since such computers will never be in demand. For instance, many automobile collectors prize 1982 DeLoreans, and these not-so-old lemons are currently appreciating at a brisk rate. Few people however, would pay very much for a Chevy Impala from the same year.

So go out and invest a few dollars in an Osborne I. People admire a lovable rogue, and I suspect that the Osborne name will always hold a certain standing in the microcomputer world. I'd also hold on to my TI 99/4A (an epoch making machine). The rare IBM Portable PC is another good bet. In twenty years, when IBM is reduced to selling add-in memory boards for the Compaq PS/2000, you'll be able to tell people that your Portable PC was the machine that started the downfall of the once omnipotent company.

As I figure it, the $5 million you'll make by selling your TI 99/4A to some 21st-century trendy type will be more than enough to wipe out your family's debts for generations to come. That is, if you are willing to part with it.

I've already got a basement full of old obsolete PCjrs. I hope you do too.

July 1987

NEW FROM CORCOMP
TI/IBM DISK COPIER

CorComp has announced a software program in module form capable of transferring ASCII files from TI to IBM/MSDOS format and back to TI-Format if you have 2 disk drives and a CorComp disk controller. It is listed in the latest TENEX catalog for $59.95. According to CorComp, you simply install the cartridge into the TI-99/4A, insert an IBM formatted diskette into Drive 1 and a TI formatted diskette into Drive 2.

To copy from the IBM to the TI, simply select option 1 from the main menu and a directory will be displayed of the IBM diskette. Then place a C next to any file you want to copy. Next press FCTN 6 and all the selected files will be transferred to the TI diskette. You can later use any TI word processor to modify the file.

Files can also be transferred from the TI to the IBM diskette by selecting option 2 from the main menu and placing a C in front of any file you can to copy. Then press FCTN 6 to start the transfer.

The main requirement for the IBM text data is that it MUST be in the ASCII format.

XB:BUG, A Review

by Scott Darling, (C)opyright 1987

XB:BUG is written by J. Peter Hoddie and Distributed by Genial Computerware XB:BUG is an unusual program. It is like DEBUG for the Editor Assembler. It can be resident in memory and called upon at any time. It allows you to follow a program as it progresses through what YOU programmed. As an experienced XB programmer, I can't tell you how many Hours I spent MANUALLY tracing, deciphering, and endless mapping of a program to see where I went wrong!! We know have such a program to take all the FUN(?) out of the old methods. XB:BUG requires XB version 110 and above, Disk, and 32K. Printer is optional tho almost essential. I also ran XB:BUG on the new Triton Superxb with no problems. XB:BUG will NOT work with Myarc XB II. as the memory locations are totally different.

The only limitation to XB:BUG is not the program but the memory limitations of XB. The program is 5K long. If you attempt to use XB:BUG on hybrid XB/AL files. You will have to remember that XB:BUG loads in Low Memory. There is a >A000 version on the distribution disk. You will be limited to and 18K program in the >A030 space. Like I mentioned, this is not a limitation of XB:BUG, but the 4A's.

To load XB:BUG, use the standard "RUN DSK1.LOAD" format or auto boot from power up. Normally you would load XB:BUG first, If, for some reason, you want to load XB:BUG after your code, there is a version on the distribution disk. After XB:BUG is loaded the READY prompt will return on the screen. Now you can load your XB program in the normal method. To activate XB:BUG you press the Control and Shift keys simultaneously, or you can do a CALL LINK("GOBUG"). Then you will be presented with the main debugger prompt. The following commands are available. Array: This allows you to inspect the contents of an string or non string array.

Breakpoints: Setting breakpoints allows you to stop the execution of the program at various points to check for the other functions of XB:BUG. It is NOT necessary to do this in all cases. But, sometimes the program may execute too fast for XB:BUG to literally catch what you want to examine.

Change: This allows you to change the value of any numeric variable. You first have to invoke a V or A commands.

Data: This gives the line number from which the next READ will get its DATA and also shows the next actual DATA item that will be read.

Files: Lists the unit number and device name associatted with each open file. The "mode" of the file was opened in is also given. Input,update,append, or output. Any data in the I/O buffer will be displayed.

Graphics: This item gives you information on 3 items. 1) Character definitions. 2) Color Definitions, and 3) Sprite status. You can manipulate all 3 items.

Kill Sound: This turns off the sound chip. You will like this after going back and forth from XB:BUG and XB.

List: Will list the program you are working on. You can set the line numbers you want to list.

Other Variable Space: This item is a beauty. It allows you to inspect variables in the main program AND also in Subprograms. This one is complicated to explain so read the manual!

Program: This supplies information about your program. Line number executing, ON ERROR line number, and OPTION BASE. Also, On BREAK, and TRACE if they are active.

Quit: Quits XB:BUG.

Subprograms: Lists all defined subprograms.

Trace: This will trace back all pending GOSUB and SUBPROGRAM returns.

Variables: List variables and functions with their current values. If there is an array, it will list the DIM. This also works on Subprograms. ?: Will list a line oS valid keystroke commands for XB:BUG.

Math functions: Allows you to perform simple calculations.

Match Function: Several of the commands in XB:BUG will prompt for a MATCH string. The one thing I will explain about this concerns the manual. It says you can use a wildcard character. Well, the printer made the Asterisk so SMALL you may miss this in the manual! I DID!! The quotation marks and the asterisk combined to make a nice inl blob to my old eyes!!

This is all of the commands. The manual documents each command far more than I have here.

In the manual are detailed instructions on how to manipulate some actual code. There are 5 sample files to play with. I would recommend that these are followed through, before attempting to work on a program that you are writing. XB:BUG is NOT a beginners program, it is very powerful and as such has the capability to destroy a program in memory! If this were to happen and you saved the resultant memory to disk.....you may be cussing for a long time! So what GRADE do I give XB:BUG? I have to give it an A in everything except Ease of Use. Why? As I stated above, this program is not for the novice. If they feel they are buying a program that will teach them XB programming or literally do it all for them. They are sadly mistaken.

FUNNELWEB FARM
EXTENDED BASIC TUTORIAL
from FUNNELWEB FARM

INTRODUCTION. In this series of notes on TI Extended Basic, we will concentrate on features which have not received due attention in UG newsletters or commercial magazines. Most programs published in these sources make little use of that most powerful feature of XB, the user defined sub-program, or of some other features of XB. The best helper is TI's Extended Basic Tutorial tape or disk. The programs in this collection are unprotected and so open for inspection, it's worth looking at the listings to see an example of how sub-programs can give an easily understood overall structure to a program.

Well, what are we going to talk about then? Intentions at the moment are to look at:

1) User-defined sub-programs
2) Prescan switch commands
3) Coding for faster running
4) Bugs in Extended Basic
5) Crunching program length
6) XB and the Peripheral Box
7) Linking in assembler routines.

Initially the discussion will be restricted to things which can be done with the console and XB only. The real virtue of the expansion system for game programming, apart from allowing longer programs, is that GPL can be shoved aside for machine code routines in the speed critical parts of the game, which are usually only a very small part of the code for a game. Even so careful attention to XB programming can often provide the necessary speed.

As an example, the speed of the puck in TEX-BOUNCE is a factor of 10 faster in the finally released version than in the first pass at coding the game.

SUB-PROGRAMS IN OVERVIEW. Every dialect of Basic, TI Extended Basic being no exception, allows the use of subroutines. Each of these is a section of code with the end market by a RETURN statement, which is entered by a GOSUB statement elsewhere in the program. When RETURN is reached, control passes back to the statement following the GOSUB. Look at the code segments:

```
290....
300 GOSUB 2000
310...
2000 CALL KEY(0,X,Y):: IF Y=1 THEN RETURN ELSE 2000
```

This simple example waits for and returns the ASCII code for a fresh keystroke, and might be called from a number of places in the program. XB provides four ways of isolating parts of a program.

1) Built-in sub-programs
2) DEF of functions
3) CALL LINK to machine code routines
4) User defined BASIC sub-programs

The first, built-in sub-programs, are already known from console Basic. The important thing is that they have recognizable names in CALL statements, and that information passes to and from the sub-programs through a well-defined list of parameters and return variables. No obscure PEEKS and POKES are needed. The price paid for the power and expressiveness of TI Basic and XB is the slowness of the GROM/GPL implementation.

DEF function is a primitive form of user defined sub-program found in almost all BASICs. Often its use is restricted to a special set of variable names, FNA,FNB,.., but TI Basic allows complete freedom in naming DEFed functions (as long as they don't clash with variable names). The "dummy" variable "X" is used as in a mathematical function, not as an array index.

```
100 DEF CUB(X)=X*X*X
```

doesn't clash with or affect a variable of the same name as "X" elsewhere in the program. "CUBE" can't then be a variable whose value is assigned any other way, but "X" can be. Though DEF does help program clarity, it executes very slowly in TI Basic, and more slowly than user defined sub-program CALLS in XB.

CALL LINK to machine code routines goes under various names in other dialects of Basic if it is provided (eg USR( ) in some). It is only available in XB when the memory expansion is attached, as the console has only 256 bytes of CPU RAM for the TMS9900.

You should have your TI Extended Basic manual handy and look through the section on SUB-programs. The discussion given is essentially correct but far too brief, and leaves too many things unsaid. From experiment and experience, I have found that things work just the way one would reasonably expect them to (this is not always so in other parts of XB). The main thing is to get into the right frame of mind for your expectations. This process is helped by figuring out, in general terms at least, just how the computer does what it does. Unfortunately most manuals avoid explanations in depth. TI's approach can fall short of the mark, so we are now going to try to do what TI chickened out of.

The user defined sub-programs feature of XB allows you to write you own sub-programs in Basic which can be CALLed up from the main program by name in the same way that the built-in ones are. Unlike the routines accessed by GOSUBs, the internal workings of a sub-program do not affect the main program except as allowed by the parameter list attached to the sub-program CALL. Unlike the built-in sub-programs which pass information in one one direction, either in or out for each parameter in the list, a user sub-program can use any one variable in the list to pass information in either direction. These sub-programs provide the programming concept known as "procedures" in other computer languages, such as Pascal, Logo, Fortran. The lack of proper "procedures" has always been the major limitation of Basic as a computer language. TI XB is one of the Basics that does provide this facility. Not all Basics, even those of very recent vintage are so civilized. You will find that with true sub-programs available, that you can't even conceive any more of how one could bear writing substantial programs without them.

The details of how procedures or sub-programs work vary from one language to another. Let's look at how XB handles sub-programs. The RUNning of any XB program goes in two steps. The first is the prescan, that interval of time after your type RUN and press ENTER, and before anything happens. During this time the XB interpreter scans through the program, checking a few things for correctness that it couldn't possibly check as the lines were entered one by one, such as there being a NEXT for each FOR. The TI Basics do only the most rudimentary syntax checking as each line is entered, and leave detailed checking until each line is executed. At the same time XB extracts the name of all variables, sets aside space for them, it sets up the procedure by which it associates

variable names with storage locations during the running of the program.

XB also recognizes which built-in subprograms are actually CALLed. How can it tell the difference between a sub-program name and a variable name? That's easy since built-in sub-program names are always preceded by CALL. This is why sub-program names are not reserved words and can also be used as variable names. This process means that the slow search through the GROM library tables is only done at pre-scan, and Basic then has its own list for each program of where to go in GROM for the GPL routine without having to conduct the GROM search every time it encounters a sub-program name while executing a program. In Command Mode the computer has no way provided to find user defined sub-program names in an XB program in memory even in BREAK status. XB also establishes the process for looking up the DATA and IMAGE statements in the program.

Well then, what does XB do with user subprograms? First of all, XB locates the subprogram names that aren't built into the language. It can do this by finding each name after a CALL or SUB statement, and then looking it up in the GROM library index of built-in sub-program names. You can run a quick check on this process by entering the one-line program.

```
100 CALL NOTHING
```

TI Basic will go out of its tiny 26K brain and halt execution with BAD NAME in 100 error message, while XB, being somewhat smarter, will try to execute line 100, but halts with a SUBPROGRAM NOT FOUND IN 100 message.

The XB manual insists that all sub-program code comes at the end of the program, with nothing but sub-programs after the first SUB statement (apart from REMarks which are ignored anyway). XB then scans and establishes new variable storage areas, starting with the variable names in the SUBxxx(parameter list), for each sub-program from SUB to SUBEND, as if it were a separate program. It seems that XB keeps only a single master list for sub-program names no matter where found, and consults them whenever the interpreter encounters a CALL during program execution. Any DATA statements are also thrown into the common data pool. Try the following little program to convince yourself:

```
100 DATA 1
110 READ X :: PRINT X :: READ X :: PRINT X
120 SUB NOTHING
130 DATA 2
140 SUBEND
```

When you RUN this program, it makes no difference that the second data item is apparently located in a sub-program. IMAGEs behave likewise. On the other hand DEFed functions, if you care to use them, are strictly confined to the particular part of the program in which they are defined, be it main or sub. During the pre-scan DEFed names are kept within the allocation process separately for each subprogram or the main program. Once again, gray a little programming experiment to illustrate the point.

```
100 DEF X=1 :: PRINT X;Y :: CALL SP(Y) :: PRINT X;Y
110 SUB SP(Z) :: DEF X=2 :: Z=X :: DEF Y=2
120 subend
```

This point is not explicitly made in the XB manual and has been the subject of misleading or incorrect comment in magazines and newsletters. A little reflection on how XB handles the details will usually clear up difficulties.

TI Basics assign nominal values to all variables mentioned in the program as part of the prescan, zero for numeric and null for strings, unlike some languages (some Basics even) which will issue an error message if an unassigned variable is presumed upon. This means that XB can't work like TI LOGO which has a rule that if it finds an undefined variable within a procedure it checks the chain of CALLing procedures until it finds a value. However, unlike Pascal which erases all the information left within a procedure when it is finished with it, XB retains from CALL to CALL the values of variables entirely contained in the sub-program. The values of variables transferred into the subprogram through the SUB parameter list will, of course, take on their newly-passed values each time the subprogram is CALLed. A little pro-gram will show the difference:

```
100 FOR I=1 TO 9 :: CALL SBPR(0):: NEXT I
110 SUB SBPR(Z):: A=A+1 :: B=B+I :: PRINT A;B
120 SUBEND
```

The first variable printed is reset to 0 each time SBPR is called, while the second, B, is incremented from its previous value each time. Array variables are stored as a whole in one place in a program, within the main program or subprograms in which the DIMension statement for the array occurs. XB doesn't tolerate attempts to redimension arrays, so information on arrays can only be passed down the chain of subprograms in one direction. Any attempt by an XB subprogram to CALL itself, either directly or indirectly from any subprogram CALLed from the first, no matter how many times removed, will result in an error. Recursive procedures, an essential part of TI LOGO, are NOT possible with the XB subprograms, since CALLing a subprogram does not set up a new private library of values.

Another simple programming experiment will demonstrate what we mean by saying that XB sets up a separate Basic program for each subprogram. RUN the following:

```
100 X=1 :: CALL SBPR :: BREAK
110 SUB SBPR :: X=2 :: BREAK :: SUBEND
```

When the program BREAKs examine the value of variable X by entering the command PRINT X, and the CONtinue to the next program BREAK, which this time will be in the main program, where you can once again examine variable values.

To summarize:

a) XB treats each subprogram as a separate program, building a distinct table of named (REFed) and DEFed variables for each.

b) ALL DATA statements are treated as being in a common pool equally accessible from all subprograms or the main program as are also IMAGE statements, CHARacters, SPRITEs, COLORs, and File specificiations.

c) All other information is passed from the CALLing main or sub-program by the parameter lists in CALL and SUB statements.

XB does not provide for declataion of common variables available on a global basis to all sub-programs as can be done in some languages.

d) Variable values confined within a sub-program are static, and preserved for the next time the sub-program is CALLed. Some languages such as Pascal delete all traces of a procedure after it has been used.

e) XB sub-programs cannot CALL themselves directly or indirectly in a closed chain. Subject to this restriction a sub-program can be CALLed from any other sub-program.

f) The MERGE command available in XB with a disk system (32K memory expansion optional) allows a library of XB subprograms to be stored on disk and incorporated as needed in other programs.

(Note: This article has appeared in several newsletters this month. We got it via the Jackson County 99ers who got it from Decatur IL, who got it from the Central Iowa group)

PROGRAMMING TECHNIQUES: MODULAR PROGRAMMING

by Tom Wynne, Puget Sound 99ers, April, 1986

I have been programming on my computer for 6 years, and I have experience in many different programming languages. I have used many techniques in trying to solve a problem, from flowcharting to pseudocode to spaghetti code. All of them will work, but some are much more time consuming and some are hard to interpret and understand. After all of that, I have found one way of programming that is efficient and easy to read. It is called modular programing. This method is greatly encouraged by schools and programmers. "Modular" means divided into modules or subroutines. This type of programming can be used in almost any computer language. In fact, the TI uses this method in their GPL (Graphics Programming Language). The GPL has subroutines built in that clear the screen, read the keyboard, do mathematics, and many other things. The efficiency of this method of programming is that instead of working on a program as a whole, you can program each small piece separately. Another advantage is that after a program is complete, it will be very easy to read and understand. Another advantage to this programming is that if you create a subroutine for a program, you can save it and not only use it for your particular program, but use it for any other program that may need the same function.

The object of modular programming is to make your main program as small as possible and keep all of the details in off in subroutines (below)> This makes the main program very readable. This program will first display the menu and get the response with CALL MENU(CH). CH is the variable the response is returned in. Line 130 will jump to the corresponding line number to CH. If CH is 1, then line 140 will be executed, or if CH is 4, lines 170 will be executed, and so on. From there a subroutine is executed. When the subroutine is finished (SUBEND, SUBEXIT) it will reurn to the main program, like a GOSUB statement, and execute the next statement (GOTO 120) and display the menue again.

You can add the subroutines to do the special things you want to do. When you make a subroutine, you must specify the vairable in the subprogram call and the subprogram like: CALL MENU(CH) and: SUB MENU(CH). If the CH is not specified, line 130 will not work. For more informaton, see page 180 in your extended BASIC manual.

```
200 SUB MENU(CH)
210 DISPLAY AT(1,9)ERASE ALL:"MAIN MENU"
220 DISPLAY AT(3,3):"1.  LOAD FILE"
230 DISPLAY AT(5,3):"2.  SAVE FILE"
240 DISPLAY AT(7,3):"3.  PRINT RECORDS"
250 DISPLAY AT(9,3):"4.  ADD RECORDS"
260 DISPLAY AT(13,3):"5.  SEARCH/UPDATE"
270 DISPLAY AT(15,3):"6.  END SESSION"
280 DISPLAY AT(20,3):"CHOICE?_ (1-6)"
290 ACCEPT AT(20,17)SIZE(1)VALIDATE("123456"):CH
300 SUBEND
400 SUB LOADFILE
```

NEW FORTH DISK
A REVIEW BY PAUL NEMMEYER
NORTHCOAST 99ERS

At our May meeting I had the opportunity to meet Wesley R. Richardson from the Kentucky Bluegrass 99 Computer Group. West gave me a disk of FORTH programs he had put together, entitled "BLUEGRASS2". This article will introduce you to the disk and hopefully encourage you to secure it.

To acquaint yourself with this whirling piece of plastic, first run the catalog which resides on Screen 89. I suggest you make a printed copy of the catalog to keep with the disk (I hope you remember how to print a screen).

Load your Forth System disk and for the first program, enter the options -TEXT and -PRINT. Next enter 22 LOAD. After loading, type PYTHAGORAS to start running. The computer will prompt you to enter a number. A series of numbers will spit out, but it beats me what they mean. Perhaps some mathematician will help us out of this quagmire.

To access another program called Fincalc, load the -FLOAT and -PRINT options, followed by 24 LOAD. A simple financial calculator will pop up. If you're intent on going into debt, you can enter the dollar value, number of payment periods, and the interest rate. You may then quickly calculate the bondage of your monthly payment .

The next program, called Doodle, offers the aspiring artist some playful fun. When tempted to mutter in psycho-babble, enter this program and doodle. This drawing program requires loading the -TEXT -GRAPH2 and -GRAPH options. Then enter 28 LOAD. Since you can use joysticks, keep the alpha lock up (you may also use the keyboard). Press the firebutton to select a line color or to turn on/off the drawing function. The following are the keyboard couriers:

R=right diagonal up	W=left diagonal up
C=right diagonal down	Z=left diagonal down
E=up	X=down
S=left	D=right
Q=on/off	

To exit program, push the firebutton over END.

One nice feature about this program is the diagonal line. On diagonal drawing, dot graphics takes over to give a clean straight line instead of the usual step effect.

Spiral is the next program. Load the -TEXT and -GRAPH options, followed by 31 LOAD. All the characters on the key-board spiral, like flashing javelins, into rectangles. Type SPIRAL to repeat the program.

How about a pleasant challenge with a puzzle called Twelve? Load the -TEXT, -GRAPH and -GRAPH1 options. 35 LOAD will bring in the program. By using a balance scale, you attempt to locate the one block out of twelve that is either heavier or lighter than the others.

The last program, called Highscore, is different. It requires the options -FILE, -COPY and -FLOAT. After 44 LOAD type HIGHSCORE. This program will make a D/V 80 files on a SSSD disk and record the name and score on both disk and screen. You might find this useful if you are writing a game program and want a highscore update.

When you finish with any of the programs and want to go to another, clear the buffers by entering FORGET IT.

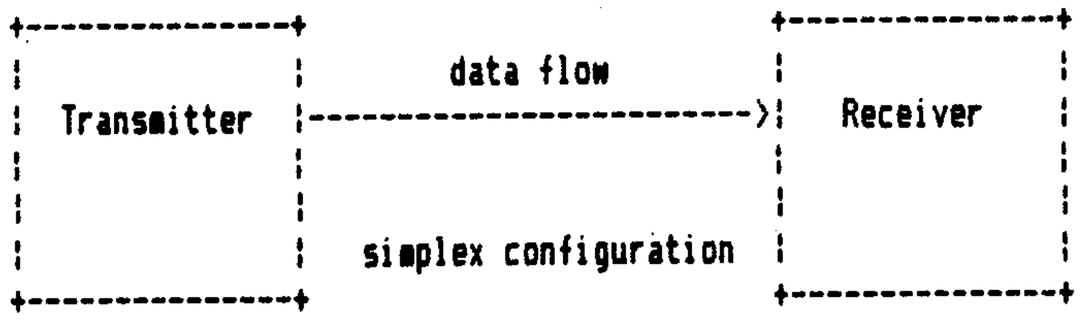
Unlike many Forth disks I've used, I had no problems with crashes or lockups. That's certainly refreshing.

SERIAL DATA TRANSMISSION

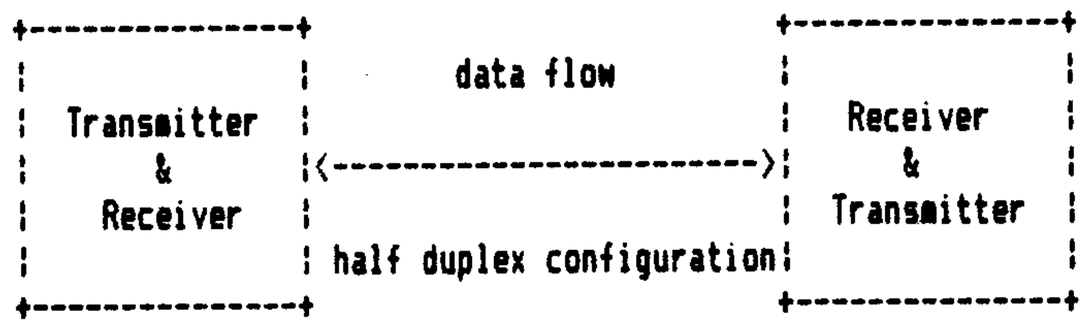
Jim Mekeel
Northcoast 99ers
Cleveland, Ohio

In serial transmission, information is transferred, on data bit at a time, between two computers.this flow of data can be one of three modes:

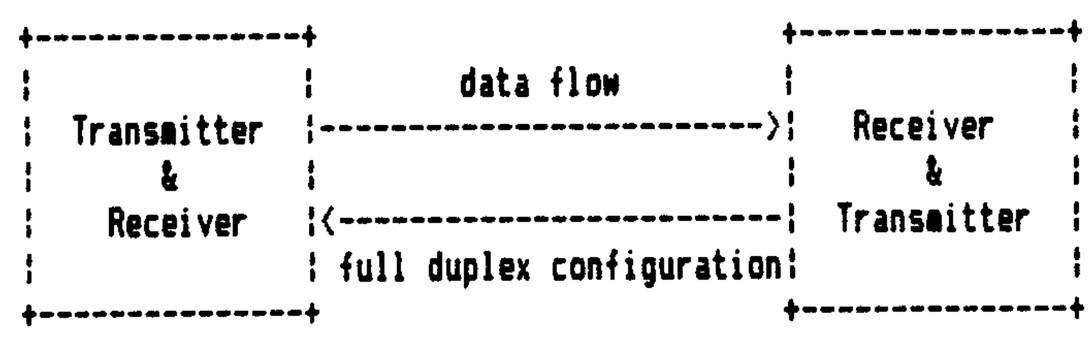
SIMPLEX - data flows in only one direction.



HALF DUPLEX - data flow is in both directions but not simultaneously.



FULL DUPLEX - data flow is in both directions and can occur simultaneously.



Data transmission can occur either synchronously or asynchronously. With synchronous transmission, faster speeds can be achieved because the two machines can be synchronized by special characters that do not consume space in each transmitted byte. However, the TI 99 computer and most other computers transmit data asynchronously so that there is no compatibility problems between different machines (TI can talk with Apple, IBM, et cetera). Within the asynchronous data stream, each character of data is transported in a binary bit frame. Each frame begins with a start bit. A low level voltage signal on the data line marks the beginning of the start bit, and the receiving computer can then begin looking for the character transmitted. The following 7 or 8 data bits comprise the binary character. (This should be set the same as the host device or program - XModem file transfers use 8 bits and TE II uses 7 bits). For error detection, an optional parity bit can mark whether the total of 0's or 1's were even or odd. A stop bit signals the end of the character.

Parity bits trap errors in the following manner: when the transmitting device frames a character, it tallies the number of binary 1's and 0's within the frame and attaches a parity bit. Then the receiving computer will count the different bits and compare the total to the parity bit sent. If a discrepancy is found a request for retransmission will be issued. The TI 99 computer uses three parity types:

ODD - Eighth data bit is logical zero if total number of logical 1's in the first dat bit is odd.

EVEN - Eighth data bit is logical zero if total number of logical 1's in the first dat bit is even.

NONE - Eighth data bit is ignored.

This parity bit must be set the same as the host computer, printer, et cetera. If you are logging on to a BBS, this and other configuration information will appear in the new user section.

Link control (or the rules for orderly transmission) is important so that data can flow without loss. The first parameter is the rate of transmission. Serial data transmission is measured in units of bits per second or bps. This is called the baud rate and on the TI can be 110, 300, 600, 1200, 2400, 4800, 9600 bps. Both machines must be operating at the same baud rate.

Also, the receiver must tell the transmitting computer when its buffer is full so that transmission can be temporarily halted. The TI and most other computers use X-ON/X-OFF. When the buffer is full, the receiving device sends an ASCII DC3 (X-OFF) signal until the buffer has cleared then it sends a ASCII DC1 (X-ON) to continue transmission.

Hopefully, this has cleared up some questions that might have come to your mind when reading the RS232 card manual. If you have further questions, send them to me and I will try to answer in the newsletter.

July 1987

ERROR ERROR ERROR

In last month's newsletter, there was an inadvertant error made in the editing of Jim Mekeel's article on file loading. To correct this, we are publishing the affected section this month:

If a PROGRAM file uses more memory than 45 sectors or uses many large arrays, some memory dedicated to other functions will need to be freed by a CALL FILES statement. This need is usually indicated by a MEMORY FULL IN xxx error message. To execute a CALL FILES statement:

type CALL FILES(1) enter
type NEW enter
then load run

The program should now run unless there is something wrong in the code.

TI SINGS

STEVE PATTERSON - New Horizons - May 87

I recently ordered a disk from TENEX called "TI SINGS". It was only $6, and I had seen a couple of the songs that it had produced, and I thought it was a pretty neat piece of software. Although it did take 2 weeks for me to get the disk (Back Order), I still thought it was worth all I had put into it.

The disk comes with six ready-to-run songs that are better than I could dream of making at this time. Plus it has a program to create the songs with its own documentation, and a program to refine the songs or put allophones into the program. The refine program also had its own documentation.

The disk itself contains some documentation explaining how to do it. Also, a couple of articles on how to use allophones in songs.

I believe the software was really well written, and I think if you like things that are a little on the strange side, you will also enjoy this disk.

NEW ADDRESS

By Steve Patterson - New Horizons - June, 1987

NO, I have not moved to Oklahoma or Idaho, but I had stumbled upon a very interesting location (Address) in VDP memory that is accessable from Extended Basic using the CALL LOAD and CALL PEEK commands. It is address -31952.

I discovered the address in a Genial Traveler program on the first disk of the diskazine. It was used to delete the program in memory and then the program ended. I began to fool around with this because of my great curiousity.

What I found, I think, is something that will benefit all that have the 32K memory expansion.

First, I would like you to try a little experiment to demonstrate the use of the Call Load.

1. Load any program into Extended Basic.
2. CALL PEEK(-31952,W,X,Y,Z)
3. Print W,X,Y,z
4. Write down those four numbers.
5. CALL LOAD(-31952,255,0,255,0)
6. Try to list or run the program.
7. Now, CALL LOAD(-31952,w,x,y,z). W-Z are numbers from call peek in 2.
8. List or run the program.
9. Then, NEW
10. Repeat 6-8.
11. Then, press FCTN "+" or "QUIT"
12. Return to XB and repeat 6-8.
13. Then, type "BYE"
14. Return to XB and Repeat 6-8.
15. Then, with widget, Hit Reset.
16. Return to XB and Repeat 6-8.
17. Turn off computer.
18. Return to XB and Repeat 6-8.

If you followed the directions, you should have been able to retrieve the program without having to reload the program itself, except for 18, I wasn't serious.

Not only can you delete the lines that are in memory an bring them back with just one statement, but you can also delete just several lines from either the end of the program or the beginning. I have been unable to delete lines rom the center of a program, but maybe there will be something in the next newsletter.

Deleting lines beginning at end

First, you must get the four numbers from the address using the call peek because most programs are not the same length and these will differ with different sized programs. Do this by a command in the edit mode:

CALL PEEK(-31952,W,X,Y,Z) :: PRINT W,X,Y,Z

Then count the number of lines you wish to delete starting at the end and moving towards the beginning. Then times that number by 4. Input this number plus "X" into the call load as "X". If the number exceeds 255, then you must add 1 to "W" and "X" becomes the amount over 255.

Example:

W=254, X=231, Y=255, Z=13

You wish to delete 8 lines from the end of the program.

8*4=32

231+32=263

263>255 by 13

So, W=255 and X=13

CALL LOAD(-31952,255,13)

If you only wished to delete 2 lines:

2*4=8

231+8=239

239<255

So, W=254 and X=239

CALL LOAD(-31952,254,239)

Deleting lines beginning at start

First, you must get the four numbers from the address using the call peek. Do this by a command in the edit mode:

CALL PEEK(-31952,W,X,Y,Z) :: PRINT W,X,Y,Z

Then count the number of lines you wish to delete starting at the first line and moving towards the end. Then times that number by 4. Input this number minus "Z" into the call load as "Z". If the number is below 0, then you must subtract 1 from "Y" and "Z" becomes 255 minus the amount under 0.

Example:

W=254, X=231, Y=255, Z=13

You wish to delete 8 lines from the start of the program.

8*4=32

13-32=-9

So, Y=254 and Z=245

CALL LOAD(-31950,254,246)

If you only wished to delete 2 lines.

2*4=8

13-8=5

5>0

So, Y=255 and Z=5

CALL LOAD(-31950,255,5)

I hope you will find this article of use, and that I can figure out a method of deleting lines from the center of a program, just not the end and beginning. If you decide to work with this CALL LOAD and find a way of deleting lines in the center of a program, be sure to let me know or write an article and let everyone know.

Both examples used in this article were based upon a program containing only 10 lines. This CALL LOAD is capable of deleting as many lines as are present in memory.

AA/TALK

DELETE FILE	TE-II				HELP				
PRINT ON	PRINT OFF	HALF/FULL DUP	BUFF ON/OFF	SAV/CLR BUFF	SEND ASCII	CATALOG	CONFIGURE	XMODEM XFER	AUTO-DIALER	QUIT

QI=I600

MODEM BAUD	PRNT ON/OFF	MODEM PARITY	MODEM PORT	PRINT PARITY	PRINT PORT	PRINT BAUD			SCR'N WIDTH	QUIT
N,=SND FILE	B=DUMP BUFF	Y=CLEAR LOG	CANCEL	WINDOW RIGHT	K=SET TIMER	TEXT COLOR	SCREEN COLOR	.=LOG ON/OFF	FREEZE TOGL	QUIT

P-TERM

TOGGL PRNTR	LOAD BUFFER	XMIT NXT LINE	XMT X/ONOFF	RESET BUFFER	DUMP BUFFER	RESTRT PROG			QUIT	
E=TOGL ECHO					SCR'N COLOR					

TE-XI

SPEAK	OUTPUT	CANCEL	TRANS	WRAP	CASE	PAGE			EXIT	
										QUIT

BASIC

DELETE	INSERT	ERASE	CLEAR	BEGIN	PROC'D	AID	REDO	BACK		QUIT

TI-WRITER

OOPS!	REFORMAT	SCREEN COLOR	NEXT PARA	DUPE LINE	LAST PARA	WORD TAB	NEW PARA	NEW PAGE	WORD WRAP	
DELETE CHAR	INSERT CHAR	DELETE LINE	ROLL DOWN	NEXT WINDOW	ROLL UP	TAB	INSERT LINE	COMMAND/ESC	LINE #'S	QUIT

MULTIPLAN

HOME	TAB	NEXT UNL CELL	FORW'D CHAR	FORW'D WORD	CHNG WINDOW	REL/ABS REF				CANCEL
LOWER RIGHT			BACK CHAR	BACK WORD		HELP	RECALC	BACKSPACE	DEL FORWARD	

RAVE_KYBD

OOPS	REFORMAT	SCREEN COLOR	NEXT PARA	DUPE LINE	LAST PARA	WORD TAB	NEW PARA	NEW PAGE	WORD WRAP	LINE #'S
LOWER RIGHT	REL/ABS REF	FORWARD CHAR	BACK CHAR	BACK WORD	FORWARD WORD	HELP	RECALCULATE	NEXT WINDOW	NXT UNL CEL	RAVE 99/101

EDIT/ASM

DELETE CHAR	INSERT CHAR	DELETE LINE	ROLL UP	NEXT SCREEN	ROLL DOWN	TAB	INSERT LINE	ESCAPE		QUIT
DELETE	INSERT	ERASE	CLEAR	BEGIN	PROCEED	AID	REDO	BACK		QUIT

FORTH

						INSERT LINE			FORTH
DELETE	INSERT	ERASE	NEXT SCREEN	NEXT WINDOW	LAST SCREEN	DEL>ENDOLIN	COPY LINE	EXIT EDITOR		

DISKO

		FCTN E=UP	FCTN X=DOWN	FCTN S=LEFT	FCTN D=RGHT				DISKO
HEX DISPLAY	ASCII DISPL	LEAVE PROGRAM	BACK 1 SCTR	RSTRT SUBPRG	FRWRD 1 SCTR		REWRITE SCTR	RSTRT PROGRM		QUIT

ADV/DIAG

E=PG/BUF UP	X=PG/BUF DN								ADV DIAGNOS
DELETE	INSERT	ERASE	NXT SCTR/TK	EXIT PROGRM	PREV SCTR/TK	HELP	RETES BADMEM	ESCAPE	SCREEN DUMP	ASCII/HEX

GRAPHX

									GRAPHX
SLOWER	FASTER	DRAW	ERASE	NO HELP	ZOOM	COLORS	LINES	CIRCLES	COPY	MENU

CHESS

NO CONTROL	KEY PRESS	REQUIRED		;=MODE CHANG		(=HELP				CHESS
TH FUNCTN	D=OFFR DRAW	J=TIME OUT	4=ERASE	E=RESIGN	P=ARANG POSN	S=SWITCH	8=REPLAY	I=BACK UP	X=SAVE GAME	==QUIT

CREDITS

COMPILED BY		ED MACHONIS					PUBLISHED IN	MICROPENDIUM	AUGUST 1986	
PRINTED	USING	"PRINTSTRIP"	WRITTEN BY	MIKE	MACHONIS	AS				

XBASHER, A Review

by Scott Darling, (C)opyright 1987

Written by Mike Dodd, Distributed by Genial Computerware, Box 183, Grafton, Ma. 01519, $10.00, All A+'s.

This program is needed by anyone and everyone!! No clarification you say?? EVERYONE has an Extended Basic program! AT LEAST one!! This program will make that one program run faster and reduce its size. GUARANTEED!! Most of us who have been around the Ti World for awhile remember what SMASH is. The BAD part about SMASH is you had to start it at night and HOPE it was done by morning!! You won't have to worry about XBASHER! Xbasher runs out of the Extended Basic environment. There are two versions available. One for TI XB and one for Myarc XB II. No mention was made of the 9640 compatibility. Probably because the 9640 will be so much faster. You can even run XBASHER on combined XB and A/L programs. Complete instructions are given on how to do this!

To run XBASHER requires that you save your Program in Merge format using the following: "OLD DSKn.filename" then "SAVE DSKn.mergename,MERGE". Then insert the XBASHER disk in drive and select XB. The disk files will determine which XB you are using and load the correct version of XBASHER. After the program has loaded, you are presented with a title screen. Next is the option screen. Which is: Shorten Variables, Crunch Lines, Remove REMS and !'s, Remove Let's, Change CALL CLEAR to DISPLAY ERASE ALL (this one alone saves you 5 Bytes!), Don't Change CALL SUB routine Digits, and Change Constants. Some of these are obvious as to what is going on.

Shorten Variables will take all your String and Nonstring Variables and shorten them to one then two character variables. There is an immense saving in memory by doing this. Tho, most people like to have a 'name' for variables. If the variable name is less than 3 characters it is no saving in memory. Its when you go over this limit that memory is being eaten away. There is also an option to print the Variable list to an output device.

Next is crunch lines. This was VERY impressive. XBASHER will crunch or combine lines together. So what about the lines that are GOTO'ed you ask?? (Well somebody will ask!!) The A/L in XBASHER keeps track of the logic flow of the program! THIS part makes the program FAR superior to SMASH!! The only bad thing about this function is that the line length of a line number is so long you may not be able to edit the new line!! Considering this is the only drawback, it is a worthwhile option! I have been able to get 8 lines of code to a line number.....so did XBASHER. Next is REMOVE REM's and !'s. Remarks are good for developing a program but are a hindrance when actually running the program. This option will delete them and restructure the resulting deletion of them. Remove LET's. PLEASE I hope everyone by now realizes the LET statement is inconsequential to programming!

Change CALL CLEAR to DISPLAY ERASE ALL. Nothing irks me more in XB programs than to see a "345 CALL CLEAR" then "350 DISPLAY AT(12,1):...". If you use "350 DISPLAY AT(12,1)ERASE ALL:" it does the same thing as CALL CLEAR and saves memory!!

Next is Don't change Sub Digits. What this option does is change the numeric constants to the characters @,\,[,],and_. This saves 2 bytes per each occurance of the variable. But, because of the nature of CALL SUB routines this may cost you MORE memory than any savings. Also, note. CALL SUB routines are like a separate XB program within a program. Consequently you can use identical variable names in CALL SUB's as in the program without any type of error received by the Basic Interpetor. Also, CALL SUBS are slower processing than GOSUB's. The only advantage is to CALL SUB's is variable variable passing!! (Are we confused yet??) Lastly is the Change Constants option. Basicaly what was said in the previuos paragraph applys to this option. EXCEPT in this environment, this option will save you memory. Don't ask me why there is a difference. Just believe me!! So much for the option list. Each option has a letter reference. By pressing that Letter toggles each option on and off. Hitting X says you like what you see on the Screen.

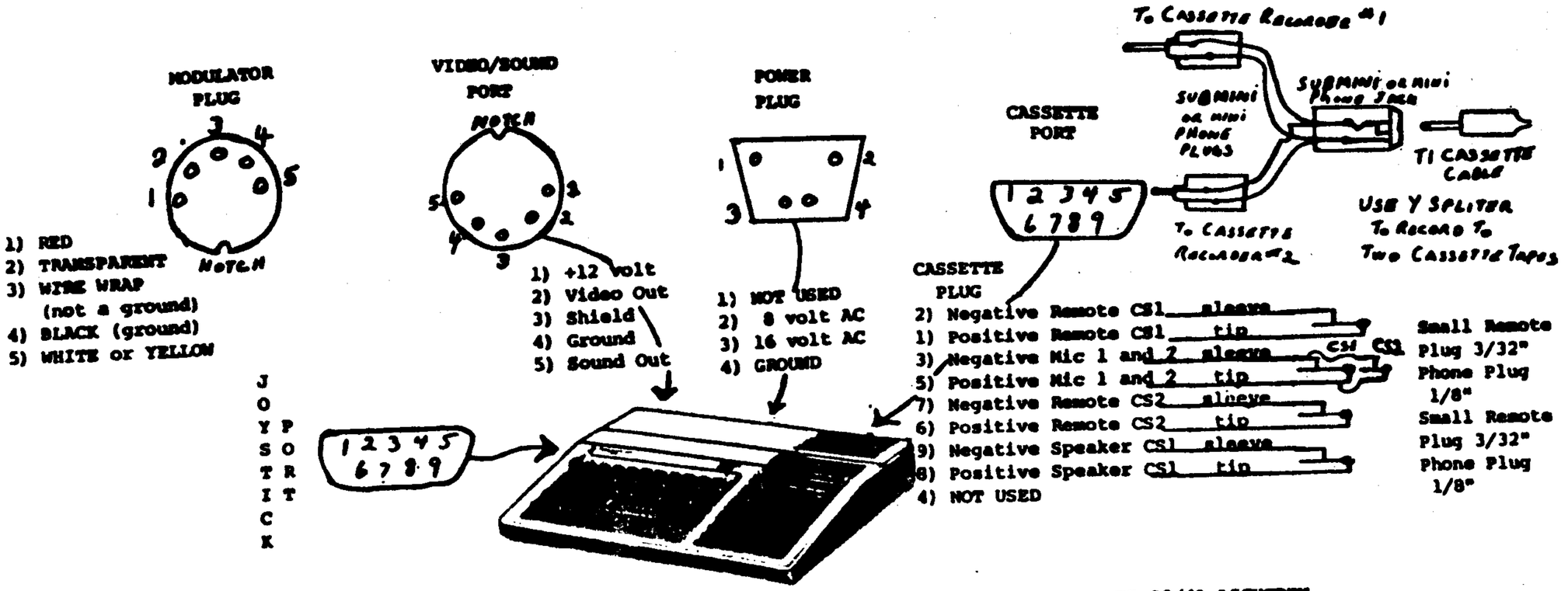
Next screen asks for the input file name. The one you saved in MERGE format and checks to see if you remembered the filename correctly. Then asks for an output name. And even provides a suggested name. Next is an output device and name for the variable listing if you selected that option. FINALLY the computer starts doing the work!! The screen will show you the status of the program. A line count, the last line number referenced by a goto, gosub statement will be shown on the screen. Xbasher makes two passes thru a program. First to make lists of variables, line numbers and other info. The second pass will write the new program to disk. How long will it take?? The size of the program involved is the ONLY factor. I ran an 11 sector file thru XBasher and it took 5 minutes to do the job. The savings were 500 bytes. Next I ran the ultimate EGO test on XBASHER. I wrote a BBS program that is 90 plus sectors long. Almost 23K in bytes. So, I ran XBASHER against it. I felt I was a decent XB programmer and there was no way XBASHER was going to save any bytes in MY program!!

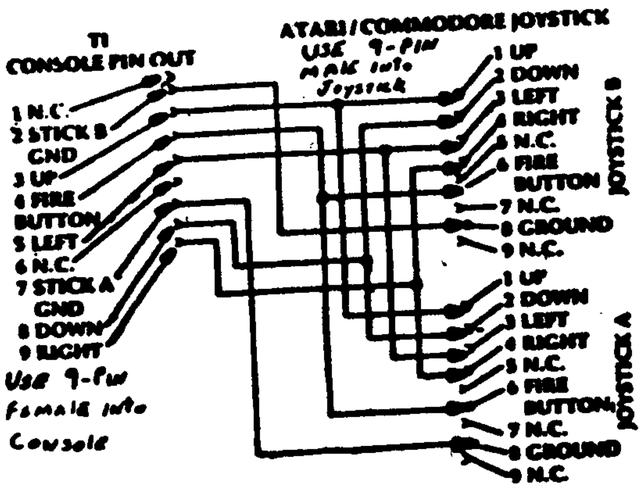
Well after about 30 minutes and my selecting ALL the options. The darn program found 200 bytes somewhere!! I'm still trying to see where it found them!!

To sum it up, Xbasher is the perfect compliment to any XB program. You only need to run it once, and save the resulting code. XBASHER will show you what XB programming is all about! There is a lot of power in that cartridge!!

W I R I N G D I A G R A M S A N D P I N P O S I T I O N S

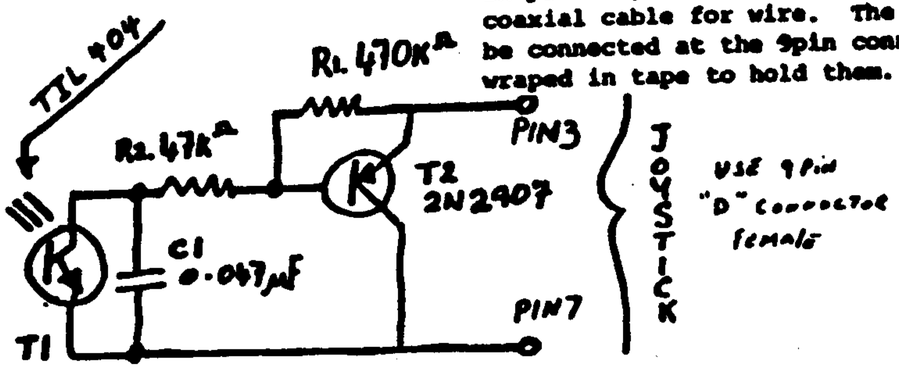
All plug and port numbers are as if you were looking straight into them.
Now you have something to use if a wire breaks or you want a weekend project.





TI-99/4A LIGHTPEN

The TIL 404 Photo-Transistor is attached to one end of the wire and inserted into a Felt-Tip or Ball Point Pen Case. It should be held in place by Silicone glue. Use RG 174/U coaxial cable for wire. The other parts can be connected at the 9pin connector end and and wraped in tape to hold them.



UNDERSTANDING TI-DOS

By Rick Felzien
West Jax 99'ERS

As I was doing my research in preparation to write a series on the use of the Advanced Diagnostics Program set from Miller's Graphics a thought occurred to me. I realized that perhaps not everyone has had the opportunity to become as familiar with the TI-DOS(Texas Instruments Disk Operating System) as I have been able to become. To use the Advanced Diagnostics Package(henceforth Diags.) effectively, one must understand a good bit about what is on the disk and what that data means to the system.

We shall begin by examining sector zero. This is where the computer tells whether the disk is initialized and keeps track of such pertinent information as sectors used/available, how many sectors are formatted, whether the disk is single or double sided, and whether it is formatted as single or double density. Let us take a look at sector zero.

```
Drive  : 1    Track  : 0
Side   : 1    Sector : 0
Byte   : 0    Display: Hex

5445535420202020202005A01244
534B202802020000000000000000
0000000000000000000000000000
0000000000000000000000000000
0300000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
000000000000000000000000FFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFF
```

This is what you would see on your screen if you used the Edit Sector (ES) command. The information is used in either byte size or word (2 byte) sized units to keep track of the data.

Bytes >0->9 are for the disk name If it is less than 10 characters long the trailing characters are blanks. (To keep things consistant hexedecimal numbers, which is how all data is used and displayed will appear with a pre-ceeding greater than character (>10)).

Bytes >0A->0B denote the number of formatted sectors, >0168=SS/SD, >02D0=DS/SD or SS/DD, and >05A0=DS/DD.

Byte >0C is the number of sectors per track, >09=9, and >12=18(for DD).

Bytes >0D->0F are the letters DSK which the controller look for to see that the disk is initialized by a TI compatible controller.

Byte >10 is used for the provected/not-protected code which was used by TI to protect some of their disks in the beginning. >20 (space char.)=Unprotected, 50 (letter P)=Protected.

Byte >11 denotes tracks per side >23=35, >28=40.

Byte >12 designates the number of sides formatted either >01 or >02.

Byte >13 tells us the formatted density either >01=SD or >02=DD.

bytes >14->55 reserved-not used-all zero's.

Bytes >38->FF are the allocation bit map. This is what tells the disk controller which sectors are used and which are free. Each byte controls 8 sectors and this is controled by the fact that each bit of the particular byte is either a 1 for used or a zero for unused. On this particular disk there are 3 sectors used.

In the allocation bit map the format is as follows: SS/SD uses bytes >38->64, SS/DD or DS/SD use bytes>65->91, and DS/DD uses bytes >92-EB, and >EC->FF are unused and are formatted to all F's. or used with all bits on. On initialization the actual bit map bytes are set with all bits off or zero's.

For instance byte >38 controls sectors 1-8 as follows:

bit nos -	7	6	5	4	3	2	1	0
sector nos -	7	6	5	4	3	2	1	0

Byte >39 would be:

bit nos -	7	6	5	4	3	2	1	0
sector nos -	15	14	13	12	11	10	9	8

And so on.

To do the mathematics for finding which sector is controlled by which bit of which byte it is easiest to convert to decimal, do the math and then convert back to hex.

```
byte no - 56 = start of 8 sector group
    >38 - >38 = 0 = sectors 0-7
     56 -  56 = 0 =    "       "

sector no / 8  + 56 = byte no    remainder * 8 = bit no
      137 / 8  + 56 =  73.125       .125   * 8 = bit 1
       34 / 8  + 56 =   4.25        .25    * 8 = bit 2 and so on.
```

When I started using Diags. and had gotten fairly familiar with the DOS, I often wondered how the Check Disk (CD) command kept track of sectors that are mapped bad. This is easy when the disk is freshly initialized as they are designated as being used in the bit map. But, I said to myself how is this done after several files are put on the disk and a good bit of the bit map is used up. How did it keep the bad sectors separated from those that were used. I looked at both a good disk and one with several bad sectors and could see no special coding, even on the track header data. Being curious I called M.G. and the people there were very courteous and helpful. When they told me how it is done I felt like an idiot for not thinking of it in the first place, so what's new. What happens is that after there are files on the disk, and if there are some bad sectors, the Diags. programming does a compare type operation. It checks the allocation bit map, and then checks the file directories for the sectors that the files occupy, and any sectors that are designated as used that are not included in the file directory information are considered to be mapped bad. neat huh?

In case any of you wondered how I got the neat Diags. display into the page without a lot of extra counting and typing, here is how. With the Output Device (OD) command you can select DSKn.filename and Diags. will dump the screen to a Display/Variable 80 disk file that can be merged into the TI-Writer file.

Drive : 1 Track : 0
Side : 1 Sector : 1
Byte : 0 Display: Hex

```
0013001A0003001400040005000 6
001E0018001C001B000700190008
00150009000A000B001D000C0016
0010000D00110002000E0012000F
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000
```

Sector 1 is the directory link and tells the disk drive where to look for the directory sectors for the files.

Although the files are placed on the disk in the order that they are saved, the link numbers are shuffled to give correct positions for the alphabetical which shows up on a catalog operation.

This data is stored in one word or two byte blocks. The first alphabetical or A program on this particular disk has it's directory link on sector 0013 or >13, but 0013 is the first number thus denoting that it is the first in alphabetical order.

When a file is deleted, it is not actually erased. The link number is removed from this sector and the bit map on sector zero is changed, but the data is still on the origional sectors and is merely overwritten as more files are added to the disk.

Drive : 1 Track : 0
Side : 1 Sector : 2
Byte : 0 Display: Hex

```
50555A5A4C4552202020000001 00
000D8E0000000000000000000000
22C0000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000
```

Sectors >2->22 are called the File headers and sometimes called the File Descriptor Blocks.

Bytes >0->9 make up the filename up to 10 characters.

Bytes >A->B are zero's and are not currently used for data.

Byte >C tells the controller the filetype. If the file is protected, the value of 8 is added to the unprotected code number.

Type	Unprotected	Protected
DIS/FIX	00	08
DIS/VAR	80	88
INT/FIX	02	0A
INT/VAR	82	8A
PROGRAM	01	09

Byte >D denotes the number of record per sector. This number equals the sector size (256 bytes) divided by the record length--(>100/>50 =>3 or 256/80 = 3). Program files always=0. DIS or INT/FIX 40 = >06, DIS or FIX 60 = 04, DIS or FIX 80 = 03 etc.

Bytes >0E->0F equal number of sectors in the file (not including the file descriptor). This is the cataloged length minus 1.

Byte >10 is called the end of file offset. For variable length files and programs this byte lets us know the number of bytes in the last sector of the file are used. It also indicates which byte number of the EOF marker. For fixed length files this is always >00. The last byte of the last file sector is an end sentinel- AA for programs and FF for all other type files.

Drive : 1 Track : 0
Side : 1 Sector : 6
Byte : 0 Display: Hex

```
44454D4F2D312020202000008803
0006C2500600000000000000000
2C3000915000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000
```

Byte >11 gives the logical record length. FIX or VAR 40=>28, 80=>50, 163=>A3, and 254=>FE.

Bytes >12->13 are the number of fixed length files or else the number of sectors in variable length files and are not used by programs. The bytes of this two byte block are reversed so that >0500 is actually >0005.

Bytes >14-1B are all zero's and are not used (reserved for future use). Does this possibly sound familiar?

Bytes >1C----These keep track of the blocks of sectors that the file actually occupies on the disk. This is done in 3 byte blocks and are not read as they appear in the block. Nybbles 4,1,&2 are the beginning sector and Nybbles 5,6,& 3 are the number of sectors occupied by that block of the file.

The following is an example of how to read these bytes in the case of a badly fractured file which is in five segments on the disk. This does not usually happen, but will hopefully show you how to read this block of data effectively.

Sector address (hex)	address contents (nybble) 12 34 56	start sector (nyb) 412	additional & prior sectors (nyb) 563	logical end sector	size of block of sectors	Subtot.
1C 1D 1E	23 30 00	023	003	026	4	4
1F 20 21	31 40 00	031	004	031	1	5
22 23 24	58 50 00	058	005	058	1	6
25 26 27	5A 10 01	05A	011	065	12	18
28 29 2A	67 60 01	067	016	068	5	23
2B 2C 2D	B4 80 01	0B4	018	0B5	2	25

Total data sectors = 25
Directory sector + 1
Cataloged sectors = 26

Hopefully the information contained in this article will be of use to new and old users alike. Usually the information set forth here must be accumilated from several sources. Hopefully it will be of value due to the fact that it is all contained on a few pages here.

BASICS OF PROGRAMMING
RICK FELZIEN -WEST JAX 99ERS -MARCH 87

There are two basic languages made to be used by the beginner. The console has Basic built in, and it is a good language for learning the basics of programming. The other is Extended Basic, which contains the commands and functions of console Basic besides having many functions and commands added to give more power and versatility.

There are some basic steps that need to be taken to write a useful program. Many think programming is a special gift or talent which only a select few people possess. Not true. What makes a good programmer is the ability to take a problem or task and write a set of instructions and commands to make the computer solve the problem or perform the task at hand.

There is a definite flow of events which must occur in sequence in order to end up with a solution to the current problem. One of the first steps is to identify and define the problem as briefly as possible. The next thing that we must do is outline the solution as to the best approach to the solution. The word "Algorithm" is often used when speaking of programming. This word refers to a specific method of solving a certain kind of problem. If you were to ask a dozen programmers to solve a given problem, you would probably get a dozen different algorithms depending on that programmer's approach. In other words, there are no hard and fast rules as to the way a problem can be solved with your computer.

The next step is to write the program. One of the things I use when stumped by a particular area of the process of problem solving, is to look at other programs for segments which can help in my final solution. One of my favorite sources are the Learning Basic and the Teach Yourself Extended Basic lesson programs. Many routines to do some of the things you can want are contained here.

I don't know of many programmers that haven't had at least a few "bugs" in their programs in the beginning. Generally these are syntax errors caused by typing errors and are usually easy to find. There are, however, some that will throw you for a loop until you find them.

The "debugging" process is somethings a lengthy and complicated task in itself.

Suppose we have a simple task such as adding two numbers and obtaining the result. We must first assign the given values, perform the necessary mathematical operation, and then be able to see the results of that operation. The following is but one approach.

```
100 REM * ADD TWO NUMBERS AND PLACE RESULT ON SCREEN
110 INPUT A,B
120 LET C=A+B
130 PRINT C
140 END
```

Line 100 is a remark statement. The computer ignores all information contained in a REM statement, but it is essential to the documentation of the program. It allows you and others to see what a given set of commands or statements do in any given instance. Line 110 allows you to input the two values and assigns the values to two variables named A and B. Line 120 adds the two numbers and places the result in a variable named C. The LET statement is used for convenience mostly, but is not necessary. Line 130 tells the computer that the program has ended.

Another task you may wish to accomplish is the squaring of a set of numbers. This can be done as follows:

```
100 REM * SQUARE THE NUMBERS AND PRINT RESULTS *
100 N=N+1
120 S=N^2
130 PRINT N,S
140 IF N<99 THEN 110
150 END
```

Line 100 is the familiar REM statement. Line 110 adds one to the value of the variable N. Line 120 Squares the number N and assigns the result of the variable S. Line 130 prints to the screen the value of N and it's square S. Line 140 tells the computer that if the value of N is less than 99, then repeat the process on the next number determined by N=N+1. This value would be the highest number you wish to have squared by the program. Line 150 is the old END statement to stop execution. On the TI the end statement is not necessary but is good practice. Note that in lines 110 and 120 we omitted the LET statement.

EXCHANGE RATES FROM TI
CARL PRETZ - NORTHCOAST 99ERS

In a call to TI-CARES (1-800-842-2737), I received the following quotes for exchanges by TI for broken TI equipment. Be sure to make a note of your Serial Number. As of the call date, the following rebuilt equipment was available. Prices and conditions are subject to change without notice. You should call for availabilty. The equipment carries a 6-month warranty.

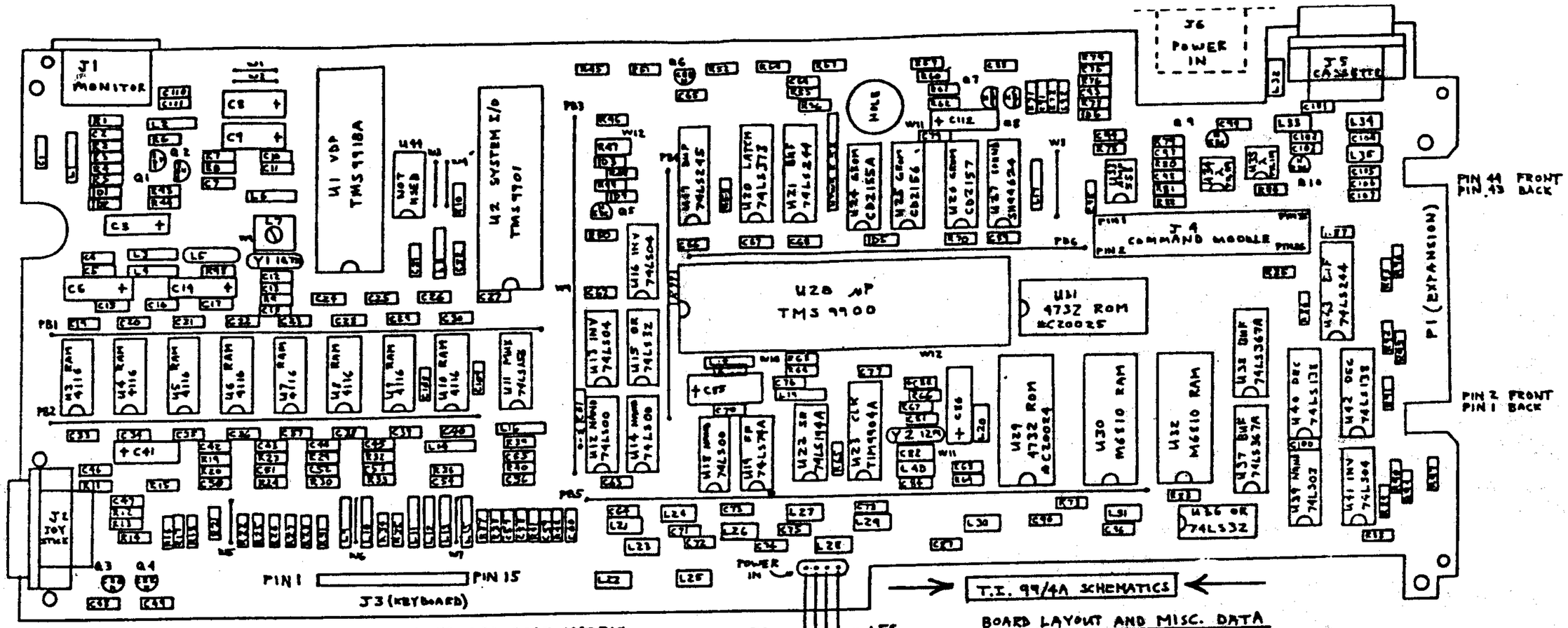
CONSOLE	$30.00 + $6.00	S&H	
P-BOX	55.00 + 6.00	"	- BARE
P-CARD	25.75 + 3.00	"	
EXTERNAL DISK DRIVE	80.00 + 6.00	"	
INTERNAL DISK DRIVE	60.50 + 6.00	"	
32K	44.00 + 6.00	"	
RS232	33.00 + 6.00	"	
DISK CONTROLLER	44.00 + 6.00	"	
SPEECH SYNTHESIZER	30.00 + 6.00	"	

TEXAS INTRUMENTS, INC. REPAIR CENTER
2305 N. UNIVERSITY AVENUE
LUBBOCK, TX 79408

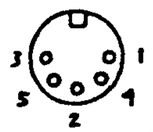
CS6D CATALOG DISK/USER DISKS #5 AND #6 RELEASES

Dave Rose of CS6D fame has announced the release of 2 more user disks, plus a catalog program which will print out a graphic catalog of CS6D fonts and pictures. It will also let you print a cross reference report of ALL CS6D large and small GRAPHICs, MESSAGE and DOCUPRINT FONTs, TI-Artist INSTANCEs, FONTs, and PICTURE files. User disks 5 and 6 will have approximately 25 large pictures, 95 small graphics, 35 message fonts and 8 docuprint fonts. The catalog program disk if $6.95 and UD #5 & #6 are $10.95 each. For anyone purchasing the 2 new user disks between June 15, 1987 and August 1, 1987, the CS6D catalog program will be included free of charge. Shipping begins July 10, 1987. Please specify type of printer. All other CS6D products should still be purchased from Texaments, or your local dealer.

DAVE ROSE
2781 Resor Road
Fairfield, OH 45014-5053

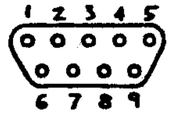


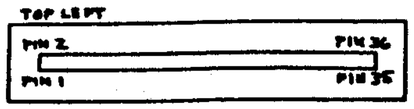
J1



VIEWED FROM OUTSIDE OF CONSOLE

J2 & J5

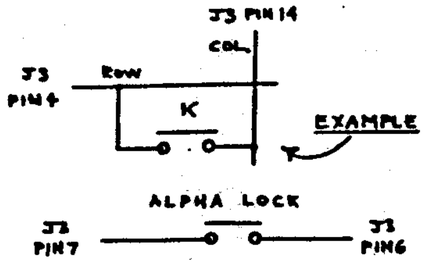




COMMAND MODULE PORT (J4A) VIA RIGHT ANGLE ADAPTER

KEYBOARD SCAN MATRIX (J3 PINOUT)

COL / ROW	P8	P13	P14	P15	P9	P12
P7	1	2	3	4	5	FCTN
P2	Ø	9	8	7	6	
P10	Q	W	E	R	T	CTRL
P1	P	O	I	U	Y	ENTER
P3	A	S	D	F	G	SHIFT
P4	;	L	K	J	H	SPACE
P11	Z	X	C	V	B	
P5	/	>	<	M	N	=



COPY DETECTOR BYE LOCATIONS		
A	1 — 8	M87-
A	9 — 16	D89-

WE	U12-3 U36-5 U27-5 J4-32 U43-11
KWR	U28-61 U32-16 U30-16 U16-5
$\overline{CS3}$	U23-7 U2-10 U43-8
$\overline{RST}$	U23-4 U28-6 U41-3 U1-34 U2-1
$\overline{CSR}$	U40-13 U1-15 U12-5
$\overline{CSW}$	U36-6 U1-14
$\overline{GCS}$	U41-12 J4-21 U24-10 U25-10 U26-10
$\overline{DBIN}$	U16-2 U16-11 U17-11 U14-9 U18-9
$\overline{RAMA}$	U36-11 U36-10 U2-5
$\overline{LOAD}$	P1-13 U28-4
$\overline{INTP}$	P1-4 U2-17
IAQB	U15-6 P1-41
$\overline{RAMCS}$	U36-8 U32-11 U30-11 U12-12
MRDY	U39-3 P1-12 U27-4 U19-12
ERDY	U24-15 U25-15 U26-15 J4-21 U41-1
$\overline{ROMCS}$	U42-15 U31-20 U29-20 U12-15

$\overline{SNDCS}$	U40-14 U27-6
CRUIN	U28-31 U2-4 P1-33 J4-6
CRUOUT	U28-30 U15-9 U2-2
CRUCLK	U28-60 U2-3
$\overline{CRUCLK}$	U16-12 J4-4 U43-15
$\overline{BLKSEL}$	U42-11 U40-4 U36-9
$\overline{MEMEN}$	U15-3 U15-12 U13-3 P1-32 U42-4
AIS/CRUOUT	U18-11 J4-8 U43-13
$\overline{DELAYEDRST}$	U41-6 P1-3
GCLK	U1-37 U24-13 U25-13 U26-13 U27-11 J4-27
SAIS	U16-8 U22-2 U20-11 U14-10 U12-9 U40-5
DBINA	U16-10 U14-4 J4-25 U41-8 U39-4 U24-12 U25-12 U26-12

SIGNAL TRACKING DIRECTORY

Improved Video

by, Bob Lawson

In my travels through the Texas Instruments Manuals, specifically the TMS-9918,28,29 Manual, I read,"The load resistor (RL, pin 36 to ground) defines the sharpness of the edges on the video signals. A lower resistor value gives faster fall times and a sharper picture." Hmmm! I don't remember any 330 ohm resistors.

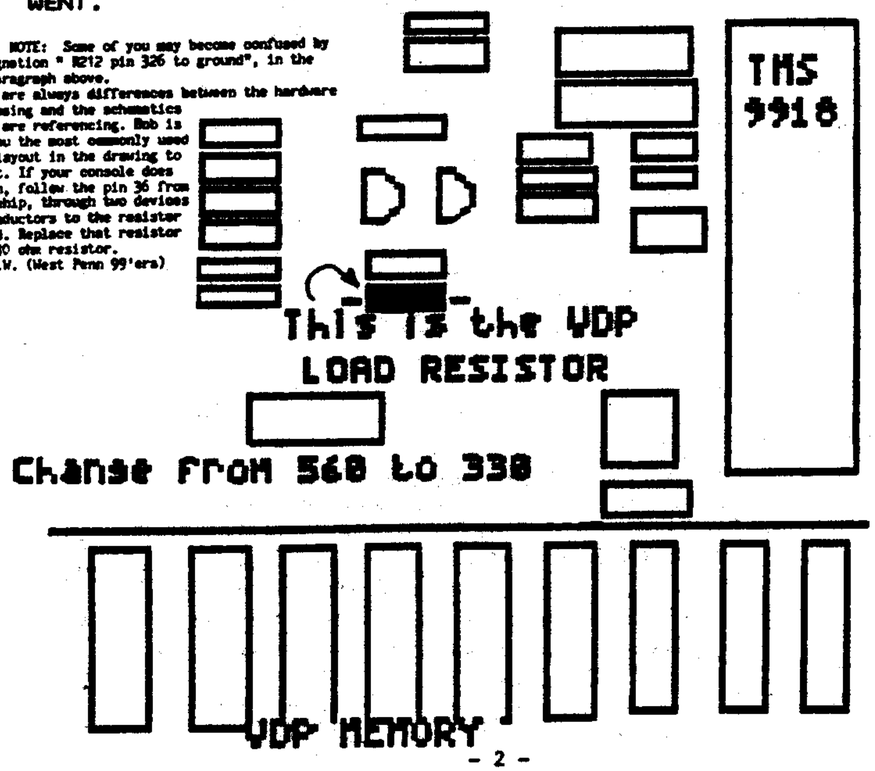
Well, I pulled out the "TI Console and Peripheral Manual, and sure enough, R212 pin 36 to ground was 560 ohms per the schematic. The next step was to check out a console, and well you guessed it, R212 was 560 ohms, not 330 ohms as recommended in the TI Manual!

Next step was to try some different value resistors, 330 ohms seems to be about the best common value resistor to use. I wonder why TI chose to use 560 ohms. I did find one old TI Manual which recommended 390 ohms (1979), but they're sometimes hard to find in 1/4 watt. This 30 cent change gives about a 40%, that's right, I said 40% improvement in the picture. The improvement is so good, you'll wonder where the WHITE SHADOWS WENT.

EDITOR'S NOTE: Some of you may become confused by the designation " R212 pin 326 to ground", in the second paragraph above.

There are always differences between the hardware you are using and the schematics that you are referencing. Bob is giving you the most commonly used console layout in the drawing to the right. If your console does not match, follow the pin 36 from the VDP chip, through two devices called inductors to the resistor to ground. Replace that resistor with a 330 ohm resistor.

J.F.W. (West Penn 99'ers)



CLEVELAND AREA 99/4A USER 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 !!