TI 99/4A
eneve

Doctor Ron Albright's ORPHAN SURVIVAL HANDBOOK

Containing:

Helpful Recipes, Assertions, Advice and other NOSTRUMS
For owners and other custodians of the . . .

TI 99/4A & Geneve

PUBLISHERS STATEMENT

As announced, Dr. Ron Albright's Orphan Survival Handbook contains over 200 pages of excellent material, much of it original work, or material not submitted to user group newsletters. Dr. Albright sifted through mountains of worthy material, selecting the best, for this continuing work.

However, neither the skill of the writer, nor Dr. Albright's deft judgement made the final decisions on what went into the book and what stayed out. The publishers, Disk Only Software, did. Therefore, some material originallly described as being contained in this publication is missing. It is merely our ability to deliver a reasonably readable product using the reproduction technique available.

With all that said, lets talk about the advantages of the medium we have chosen to place before you. On the first book, Orphan Chronicles, Dr. Albright received some comment that material just submitted to printing had become out of date. The 99er and the Geneve user is an independent sort, seeking out his information where he can find it. Then why not produce a book, drilled for use in a three ring binder? This dictated a slightly more expensive method than that used to print two thousand bound books. Cheaper per copy, bound books would be closed and a hindrance to revision until a mountain of books had been sold. Thus our format.

Next comes an invitation for easily copyable material. Authors are invited to send disks or upload their material to one of the telecommunications services. Many no longer charge while uploading. Or forward it to Dr. Ron Albright at the address below. Return of your disks will be attempted, but not guarenteed. Return postage would be helpful, of course.

Thanks to all for your support in this effort. If you are not purchasing this book direct from Disk Only Software, and wish to be personally notified of future updates, please write the address below.

Jeff Guide
Jim Horn
Publishers
March 25, 1987

First Printing, January, 1987
Second Printing, February, 1987
Third Printing, March, 1987

Fourth Printing, April, 1987

Published By:

Disk Only Software
P.O. Box 244
Lorton, Virginia 22079

or call

1-800-446-4462. At the tone, enter **897335** for recorded order message. Touchtone phone is required
Alternate is **(301) 369-1339.** No Touchtone is required.
Delphi: TELEDATA—CompuServe: 74405,1207—MCI: TDG—TELEX: 6501106897 MCI

The Orphan's Survival Manual

Foreword

As it said in the flyer for this manual, it was both easier and harder to do than the "Orphan Chronicles" were. Easier because almost everything in this manual was already written - by you, the user groups of the TI community. The only hard part (and it was tough) was selecting what to include. This manual could have been easily 300 or more pages. The amount of information available is a tribute to you, the TI users.

I wanted to personally thank those who helped get this manual put together. All the authors and user groups who helped me assimilate this mass of information have been of immense assistance. In particular, Kent Sheets of the Northwest Ohio Users Group, Art Byers of the Central Westchester 99ers, and Terrie Masters of the LA 99ers Users Group. Quite literally, this manual could not have been done without the help of these fine folks.

The authors who wrote new material for this manual deserve a special mention. They did it for the sheer love of the TI community and the desire to share their wealth of information with you. Warren Agee, Jerry Coffey, Scott Darling, Jeff Guide, Howie Rosenberg, Barry Traver, and Jonathan Zittrain all wrote along their lines of expertise for this manual. I will always be deeply appreciative.

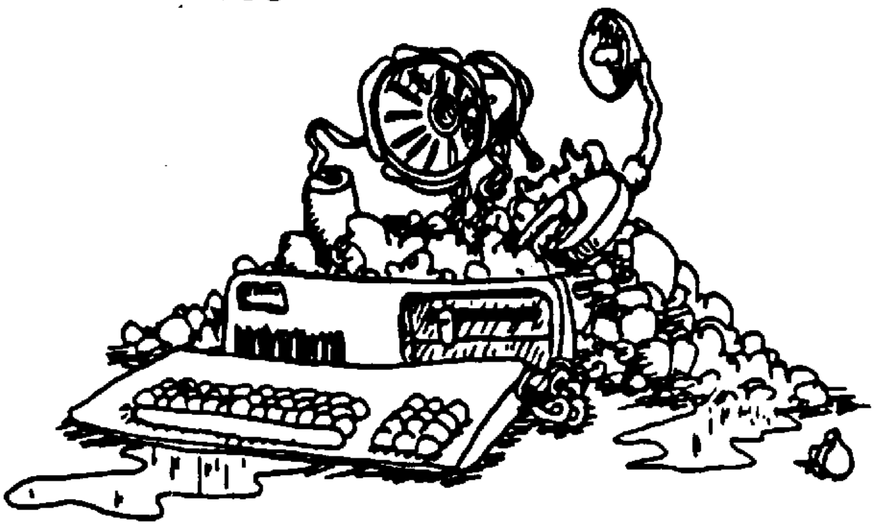
I want to remind all of you who brought this manual to send in the registration card to User Network 99. Terrie Masters has great plans for this organization and I support her efforts to unite the user groups and the non-affiliated TI users for the distribution of information. I hope to be able to work with her in making updates available for this manual. To get the information to you, she will need your address. Mail the card in right away.

Dedication? Who could this manual be dedicated to other than YOU - the TI user. Struggling against the odds. Inventing. Ingenious. Sharing. Thanks to you, we are all alive and well.

Ron Albright, Editor

Our Thanks to . . .

Bayou Byte Newsletter
ROM Newsletter—Users Group of Orange County
Manasota Users Group
Topics—LA 99'ers
Call Sounds—Central Westchester 99'ers
Northcoast 99'ers
99'er News—Chicago & Wills County Illinois
North New Jersey 99'ers Group Newsletter
MANNERS News—Mid Atlantic 99'ers, Bill Whitmore, Editor
NH99'er User Group
A9CUG
GEnie, Rockville, Maryland
OH-MI-TI
99'er NEWS
Northwest Ohio 99'er NEWS
HOCUS Newsletter
Central Texas 99/4A User Group
Delaware Valley User Group
Front Ranger, Colorado Springs, Colorado
Western Penn-WPUG



"we CAN do it!
Principal Survival Principles For Life in the Orphanage"

by Barry Traver
Genial Computerware

The TI-99/4A is alive and well and living in ... Philadelphia, Boston, Chicago, Los Angeles, Seattle, Ottowa, Washington, D.C., and elsewhere! In many respects, more exciting things have happened recently for the benefit of TI'ers than took place while Texas Instruments was still officially supporting the computer.

Life in the orphanage, however, is different in many respects. Losing a parent sometimes draws the family closer together, and TI'ers know the reality of that, but - now that we have to "make it on our own" - there are important computer "rules of life" to put into practice so as to continue to exist as a thriving community or family. It will take a cooperative effort, but "we CAN do it" if we remember three "principal survival principles": Cottaging, Archiving, and Networking. (If you remember the phrase, "we CAN do it", the word CAN provides you with a memory key for remembering these three principles of operation: _C_ottaging, _A_rchiving, and _N_etworking.) These three will be described one by one, since the meaning of the terms by themselves may not be immediately evident.

First, we need some background to understand what is meant by Cottaging. Whatever people may call it, we are in the midst of a third major cultural revolution. The first major economic orientation was the agrarian or agriculture-oriented society. Simply put, this means that many people worked on their own farms, and those who didn't usually also had home-oriented or family-oriented businesses. To put it one way, a person's workbase was his own home or cottage.

Then came along the Industrial Revolution. What this meant was that workers often did not work in, at, or near their own homes, but in large factories or other large business places of operation often some distance away. In order to earn a living, people had to leave home and become commuters. A man's home may have been his castle, but it was no longer his place of work. He no longer worked "for himself," but for other people, and often a large, multi-million-dollar company.

If you aren't interested in a personal interpretation of sociological history, you can skip this paragraph and the next, but some may find it of interest. As I see it, one of the unfortunate effects of this revolution was its contribution to the weakening of the family, since - particularly if he was a commuter - "Dad" often only got to see his family a few hours each night. In addition, whatever good effects it may have had, the simultaneous re-orientation away from home-based schooling to total classroom schooling was another sociological change that weakened the coherence of the family.

Most people probably just accept today as a "given" that the "normal" way for things to operate is that Dad works away from home and the kids go to school, but such was not the normal practice for millennia! It's a comparatively recent development that has only been in place in our country for a couple hundred years, and - although some people may experience "future shock" become of the "new" choices becoming available - there is evidence that both of those commonplace suppositions are being challenged more and more first by the increasing appearance of "cottage" industries, and the second by the growing home schooling movement.

Here we come to the third revolution. We now live in a Computer Age, whether that is your preferred term for it or not. The computer revolution is producing "cottage" workers again. There are two reasons for this. First, even for the person who is working for a large company, if it is computer work, he can do it at home, communicating with his company's computer via modem. Second, millions of dollars of resources are not necessary, just a good product. Thus "mom-and-pop" outfits can produce (and have produced) superior merchandise to that released by billion-dollar companies.

The point of application here is that we don't need Texas Instruments to survive, if we recognize - and support - the resources available from such "cottage" operations: individuals, families, or small companies who can provide (and are providing) items for the TI-99/4A that TI never provided (and perhaps never would have provided, even if TI had continued to support the TI 99/4A).

Let's look at two examples. (1) TI gave us the Terminal Emulator II. (Before that they gave us a Terminal Emulator I, which was even worse!) That was not "cottage" industry: that was what a mammoth company was able to produce. Well, now we have FAST-TERM (Paul Charlton), PTERM (Richard Bryant), 4A/TALK (Thomas Frerichs and Michael Holmes), and MASS TRANSFER (Stuart Olson), just to name a few terminal emulators that offer much more than TI's TE2 did: 1200 baud operation, XMODEM transfers, large capture buffers, and much more. (2) TI gave us a 32K RAM memory card. That was it. But 128K cards (or better!) have been made available to us by Foundation, Horizon, Myarc, Mechatronics, CorComp, and others. Do you see why some people believe that we may actually better off now that we are not dependent upon Texas Instruments but are looking to "cottage" companies to support us?

The second principal principle is Archiving. The reference here is _not_ to my ARCHIVER program - used for packing and unpacking related groups of files on disk - but just to collecting TI material in general. Why didn't I call this principle "Collecting" then? Well, this article is based on a talk I gave for the 1986 Boston TI Fayuh, which was before my ARCHIVER program made its reputation. And, besides, Collecting would mess up the "CAN" memory aid, so let's keep with the term "Archiving" here.

The idea here is that we make sure that we collect, preserve, and make available what has already been done. Although there may some benefits in re-doing certain things, often it is wasteful of time and effort for people to write new programs from scratch where public domain programs already exist that perform the same functions (and perhaps more efficiently). (Even worse, people who aren't programmers may just "go without" because programs that they need have just gotten lost.)

Two types of items actually need to be archived or collected: software and information. It especially takes a deliberate effort to preserve the latter, because often the information appears where preservation is not automatic: user group newsletters, notes on local TI BBS's, even informal conversation. Some individuals in the TI community have done some useful deliberate effort to preserve the archiving - especially Guy-Stefan Romano of AMNION Helpline - but a more organized effort is needed here. AMNION and some others have done commendably, but _all_ of us must to a certain extent become "archivists" for the sake of the TI community.

Here's a _caveat_ _non_-_emptor_ (excuse my bad Latin!): I am _not_ supporting the idea of collecting _pirated_ software. We will have "cottage" industries around to support us only to the extent that we ourselves support the TI community. You can (and should) personally archive original copies of copyrighted software for your own use, but that is one area of your archival library that you should not share with others. Public domain and Fairware material, on the other hand, you should both archive and distribute freely without restrictions. (And be sure to support Fairware authors, because Fairware software in not "free software" but "try before you buy" software that should be dealt with in integrity if we are to survive and thrive as an orphan community.)

The third principle is the principle of Networking, which merely means working together as an extended family. You should belong to and support at least one TI-99/4A user group, and that group may be local or not. (For example, some groups - such as Chicago, Boston, and Washington, D.C. - have members that live at a distance.) Also, if you have a modem, you should be actively involved with electronic databases, whether they be commercial, national databases (CompuServe, the Source, GEnie, Delphi) or local TI BBS's.

Since we can less and less look to Texas Instruments for specific help, we need to help one another more and more. This involves getting involved in specific activities that put us in touch with one another. In other words, we need to "plug into the Network." I've often had other users answer questions for me where Texas Instruments was of little or no assistance. That's to be expected, because _we_ are the ones who are now using out TI's on a daily basis.

User group newsletters and software/textfile libraries can be a great help, but one of the best resources is simply "Question and Answer." This can be done through user groups or through leaving messages on TI bulletin boards. In spite of what some people think, I'm not professionally trained in computer science; if I know anything, it's because I've asked lots of questions and listened to the answers, as well as listening to the conversations of others at user group meetings and on electronic bulletin boards (including especially FORUM on CompuServe, where I am currently serving as a Sysop).

(Incidentally, a more formal way of making contact with what's going on in the TI world is through subscribing to various TI-oriented publications, such as MICROpendium, SMART PROGRAMMER, and TRAVelER (a disk-based periodical), but I hope that you're doing that already.)

One other place where you can "network" or "make connections" with other users is in the various TI Faires that are taking place all across the country. In addition to Chicago, Boston, Los Angeles, and other places already mentioned in the first paragraph (actually, Philadelphia had not yet had a full-fledged Faire, but it has sponsored assembly language seminars by Mack McCormick and J. Peter Hoddie), other localities have sponsored such special specifically TI-99/4A events (e.g., Milwaukee and TICOFF in New Jersey). Here is where you can get to meet and talk in person with the "Who's Who's" of the TI world.

During the years that Texas Instruments was officially supporting the TI-99/4A, we had only _one_ such Faire: the TI-FEST in San Francisco. Now that we are orphans, however, announcements of new Faires are a regular occurrence. As an orphan, however, you will get little benefit from your TI-99/4A unless you put these three principal principles into practice: the Cottaging (i.e., realizing that individuals and small companies can put out products equal or superior to those from TI), Archiving (i.e., collecting in a systematic way what has already been done for our machine), and Networking (i.e., working together with other TI'ers). We CAN survive _and_ thrive: not merely as "orphans," but - as what we have become - as _family_!

Editor's Note: Mr. Traver is surely one who practices what he preaches. He has been an active participant in all three areas that he recommends to others. Most notably, the "Cottager" aspect. As owner of Genial Computerware (835 Green Valley Drive, Philadelphia, PA), he has produced the "TRAVelER Diskazine", first (really) "magazine-on-disk" for the TI 99/4A. This highly-acclaimed publication has been phenomenally accepted by the TI users and has brought such innovative programming techniques as the "Archiver" utility that Barry mentions above - the first file compressor and library utility for the 99/4A, among others. The announced association of J. Peter Hoddie with Genial is sure to bring about some incredible and innovative hardware and software from this "Cottage Industry". Barry also has his son, John Calvin, involved in "cottaging" as John has a business distributing disks of public-domain software. The Travers are, surely, a "third-wave family".

Orphan's Survival Manual
Table of Contents

-Appendices

C Gives Extra Power When You Can Use A Little Help

Getting Down to Basics . . .

COMPUTER MUSIC

We can write music with a list of 'CALL SOUND' statements, but that is very cumbersome and uses too much memory. It would be possible to write an entire composition using only one 'CALL SOUND' statement (with the exception of leading in or trailing notes.)

My method uses a main 'CALL SOUND' statement with all of the notes listed in 'DATA' statements. It will be easier to debug if each 'DATA' statement is one bar. Quite a few beginning programmers seem to have dataphobia, but 'DATA' statements are very easy to use. The following examples play the same five notes for one second each. Since there are only five notes, each program has five lines. If I added ten more notes, the first program would have fifteen lines but THE 'DATA' program would still be only five lines long. This memory savings would augment to an exponential factor with longer programs.

In the second example, we put the 'CALL SOUND' statement in a 'FOR NEXT' loop with the number of repetitions equalling the number of elements (notes) in the 'DATA' statement. Whenever the computer encounters a 'READ' statement, it goes off looking for some 'DATA' to read. It can be anywhere in the program. In this case, it calls the data 'N' for note. The note (466) is put in the 'CALL SOUND' statement and once a piece of 'DATA' is used it is no longer available, so the second time through the 'FOR NEXT' loop 'N' will be equal to 392, and so on. We could play the notes over again in both examples by adding a 'GOTO 10' line at the end of the program, however, in the second program we would first have to add a 'RESTORE' line before we could use the same 'DATA' a second time around.

The following program will play three notes at once and also handle different note durations using the 'DATA' statement. Look at the difference in the 'READ' statement, and notice in line 30 how the duration is changed by multiplying a single digit with the constant (500). To change the tempo, use a different constant

```
10 REM EXAMPLE ONE
20 FOR X=1 TO 5
30 READ N
40 CALL SOUND(1000,N,1)
50 NEXT X
60 DATA 466,392,330,262,349
70 RESTORE
80 GOTO 20
```

```
10 REM EXAMPLE THREE
20 CALL SOUND(1000,466,1)
30 CALL SOUND(1000,392,1)
40 CALL SOUND(1000,330,1)
50 CALL SOUND(1000,262,1)
60 CALL SOUND(1000,349,1)
70 GOTO 20
```

```
10 REM EXAMPLE TWO
20 FOR X=1 TO 5
30 READ D,A,B,C
40 CALL SOUND(D*500,A,1,B,1,C,1)
50 NEXT X
60 DATA 2,330,392,466
70 DATA 1,262,330,392
80 DATA 2,196,262,330
90 DATA 1,165,196,262
100 DATA 4,220,262,349
110 RESTORE
120 GOTO 20
```

COLOR BAR GRAPHS

This short program in TI Extended BASIC is very simple to use. You may use from 2 to 4 bars on each graph and each bar may be a different color. You are asked the maximum possible value of each bar. In other words, what is 100% performance? If the goal this year for the Acme Computer Company is to have each of three representatives produce 10,000 units, then maximum performance for each representative would be 10,000. Minimum performance, of course, would be zero.

The value of each bar is the relative value of each in regard to the maximum goal. In the example mentioned, producing 7500 units would give a representative 75% performance, so his bar would extend ¾ way across the screen. The title of the graph will appear at the top of the screen, and the title caption for each bar appears directly above each bar. The maximum and minimum values appear at the lower corners of the screen.

If you are doing an audio-visual presentation and need some color bar graphs in a hurry, this program could be a big help. By photographing the screen of your monitor with a single-lens reflex camera and slide film, you could use the graphs in your slide shows. Or, by sending the video signal from your computer to a video recorder, you could tape the images for incorporation into a video presentation.

```
100 CALL CLEAR
110 INPUT "HOW MANY BARS? (2-4):"
:B :: IF B<2 OR B>4 THEN 110
120 PRINT "TITLE OF GRAPH:":"(28
CHAR.MAX)" :: INPUT T$ :: IF LEN(
T$)>28 THEN 12
0
130 PRINT "MAX.POSSIBLE VALUE OF
BARS (100%):" :: INPUT MV :: IF M
V<0 THEN 130
140 FOR I=1 TO B
150 PRINT "TITLE OF BAR#";I;":":"
(28 CHAR MAX.)" :: INPUT TB$(I)::
 IF LEN(TB$(I)
)>28 THEN 150
160 PRINT " 2-BLACK        3-MED G
REEN":" 4-LT GREEN     5-DK BLUE":
" 6-LT BLUE
  7-DK RED"
170 PRINT " 8-CYAN         9-MED R
ED":"10-LT RED      11-DK YELLOW":
"12-LT YELLOW
 13-DK GREEN":"14-MAGENTA     15-G
RAY"
180 PRINT "ENTER COLOR OF BAR #";
I;":" :: INPUT C(I):: IF C(I)<2 O
R C(I)>15 THEN
 180
190 PRINT "ENTER VALUE OF BAR #";
I;":" :: INPUT V(I):: IF V(I)<=0
OR V(I)>MV THE
N 190
200 REP(I)=32*(V(I)/MV):: IF REP(
I)<1 THEN REP(I)=1
210 NEXT I
220 CALL CLEAR :: CALL SCREEN(16)
230 P$="FFFFFFFFFFFFFFFF" :: CC=9
6
240 FOR I=1 TO B :: CALL CHAR(CC,
P$):: CALL COLOR(I+8,C(I),I):: CC
=CC+8 :: NEXT
I
250 DISPLAY AT(2,15-LEN(T$)/2):T$
260 FOR I=1 TO B :: DISPLAY AT(5*
I,1):TB$(I):: NEXT I
270 CC=96
280 FOR I=1 TO B :: CALL HCHAR(5*
I+1,1,CC,REP(I)):: CALL HCHAR(5*I
+2,1,CC,REP(I)
):: CC=CC+8 :: NEXT I
290 DISPLAY AT(24,1):"0" :: DISPL
AY AT(24,28-LEN(STR$(MV))):MV
300 CALL KEY(0,KEY,STATUS)
310 IF STATUS=0 THEN 300
320 PRINT "ANOTHER GRAPH? (Y/N)"
:: INPUT Y$ :: IF Y$="Y" OR Y$="y
" THEN 100
330 STOP
```

PROGRAMMING TIPS

The time required to test and debug a program usually exceeds the tin it take to rewrite the program. Several methods are available whic will make this job easier by preventing or trapping errors which occur while a program is running. No one wants to spend time entering data and then lose it due to a program error.
One of the easiest ways to reduce errors when writing a program is to use the Extended BASIC statement ACCEPT instead of the more common INPUT statement. Using ACCEPT will require us to give up the convenience of the included prompt option available with INPUT, but will allow us to VALIDATE the keyboard input. There are several options available with VALIDATE. ACCEPT VALIDATE [UALPHA];A$ permits entry of any uppercase alphabetic character. Substituting DIGIT for UALPHA permits 0 through 9, and using NUMERIC will permit those numbers as well as: . + - and E.

Many programs ask the user a question to be answered by "YES" or "NO". The program lines could be written:

```
400 INPUT "DO YOU WANT A HARDCOPY? [Y/N]" = A$
410 IF SEG$[A$,1,1]="Y" OR SEG$[A$1,1]="N" THEN 440
420 PRINT "PLEASE RESPOND EITHER [Y]ES OR [N]O."
430 GOTO 400
440 IF SEG$[A$,1,1]="N" THEN END
```

Using another option available with ACCEPT VALIDATE, a string may entered with the characters permitted as inputs ACCEPT VALIDA ["Y,N,A"]=A$ permits only three characters to be entered as A$. Addin the SIZE option with SIZE=1, only one of the permitted characters could be entered. With these options, the previous example could be written:

```
400 PRINT "DO YOU WANT A HARDCOPY? [Y/N]":A$
410 ACCEPT VALIDATE ["Y/N] SIZE[1]:A$
420 IF A$="N" THEN END
```

With these lines in our program, pressing any key other than Y or N will result in a rude honk as will any attempt to enter a second character. Both examples will prevent the user from entering a character which the computer has not been instructed how to handle and will, therefore, reduce the possibility of an error in your program. Granted, erroneous entries for the examples given will normally result in a WARNING at the time of input; however, errors later in the program may have been prevented.

What can you do about errors which can occur late in a program? We can make use of the ON ERROR statement to trap many of these errors allowing us to recover and continue without losing data which may have already entered. When an error occurs and the program stops, it can restarted only with the RUN command. But, when RUN is entered, values of our variables are lost.

One place where errors often occur is in a program which reads DATA statements. When an attempt is made to read data past the last item in a DATA statement, the data error message appears on the screen and the program stops. An ON ERROR statement can be used to prevent this type of error. Consider the following program:

```
110 READ A :: PRINT A
120 GOTO 110
130 RESTORE
140 DATA 111, 112, 113, 114, 115
```

Running this progfam will print the numbers in the DATA statement in line 140 until 115 have been printed. After 115, the last item in the DATA statement is printed, DATA ERROR in 110 is printed on the screen and the program stops. However, if we add: 100 ON ERROR 130 and change line 130 to: 130 RESTORE :: RETURN 110, the program will run until stopped by FCTN 4 or QUIT.

Many other uses for ON ERROR can be found. Even fatal I/O errors can be trapped. To illustrate, check the following program:

```
100 ON ERROR 240
110 PRINT #1:"THIS IS A TEST"
200 ON ERROR 280
240 OPEN #1:"PIO" :: RETURN 100
280 CLOSE #1
290 END
```

The first error is created by line 110 which generates an I/O error since File No. 1 is not open. The error takes program execution to Line 240 where file No. 1 is opened as "PIO" and execution is resumed in Line 100. A second error is generated when an attempt is made to open the same file again. This is handled by Line 200 which jumps the program execution to Line 280 which closes the file.

A more practical application can be found in the following example using the CALL ERROR statement:

```
100 ON ERROR 130
110 OPEN #1:"PIO"
120 GOTO 170
130 CALL ERR(W)
140 IF W=130 THEN PRINT "ARE BOTH P.E.B.  AND PRINTER ON?"
150 PRINT "ENTER 'CON' TO CONTINUE." :: BREAK
160 RETURN 100
170 END
```

Here, the ON ERROR transfers the program to the CALL ERR statement. If W is 130 - which indicates an I/O error - the message reminding the user to turn on his P.E.B. and printer is displayed on the screen and a BREAK in Line 150 permits the necessary corrections to be made. CON will continue program execution to Line 100 for a second try.

When debugging your programs, the ON ERROR can be used to trap any error as it occurs. A CALL ERR can be used to identify the error and th program steps can be written to permit the error to be corrected and t program allowed to continue.

The CALL ERR statement has the capability of returning four values. It CALL ERR [N,N,O,P] ON ERROR and PRINT M,N,P are included in your program, most errors can be identified while the program is being debugged. For example:

```
100 ON ERROR 250
110 ! PROGRAM LINES
-
-
250 CALL ERR [M,N,O,P]
260 PRINT M,N,P
270 BREAK
280 ON ERROR 250
290 RETURN EXIT
```

Errors occurring in the program will cause execution to shift to Line 250 due to the ON ERROR 250 in Line 100. Line 250 assigns variables to M[Error Code], N[Error Type], O[Severity] and P[Line Number].

Line 260 prints the values assigned to the variables. [Severity always 9 and there is no need to print O.] Printing W gives the error code and the code number can be found in the list of error codes in t Extended BASIC manual. If the value of N, the error type, is "-1", t error occurred in a statement. P will be the line number of t statement causing the error.

A BREAK statement was included to provide an opportunity to the programmer to correct a correctable error, continue, and resume program execution at the line following the line in which the error occurred.

Once an error has been processed, it is cleared and must be executed again to handle to subsequent error. This was done in the previous examples by RETURN followed by the line number of the first ON ERROR statement. In this example, the RETURN NEXT bypassed the ON ERROR in Line 100; therefore, ON ERROR 250 is repeated in Line 280.

ON ERROR statements are similar to a GOSUB so far as a RETURN being required. Three options are available for RETURN with ON ERROR: RETURN alone will resume program execution in the statement which caused the error; RETURN NEXT causes the program to resume in the line following the line where error occurred; RETURN [Line Number] starts execution with the line number specified.

SOME BASIC THINGS . . .

ADDING HARD COPY TO PROGRAMS by George F. Steffan

...I was asked several questions about converting programs which had output only to the screen so that they would output to a printer. I also had just done such a conversion for the group library. The next day, I received a copy of the newsletter of the Wichita (Kansas)99er's Users Group [which] contained a program by Paul Yorke of Florida (no credit for original publisher given) which converted a program to use SPEECH on the TE II. I saw that this program could provide the solution to problems of this conversion.

My first thought was just to change SPEECH to RS 232 but some people would need to use PIO or different Baud rates, so I decided to allow input of the desired output device. Also, I eliminated restrictions on names for the original and new programs. I added provisions for either adding the new output device to screen display or using the output device instead of screen display.

You should use the RESEQUENCE or RES command on your program before running this program because some lines must be inserted between lines of the original program. The inserted lines are numbered 5 higher than the line from which they are derived. Therefore, resequencing is not necessary if the gap between lines is always more than 5.

If your copy of the original program is exactly the same as the old copy saved with the MERGE command, you may then speed up final recovery of the program by using "OLD and OLD PROGRAM NAME", then "MERGE and NEW FILE NAME".

This program adds " #1: " to any PRINT statements in the source program. Therefore, DO NOT USE IT on a program which already has opened a file for output and contains "PRINT #" statements.

```
100 REM ADDPRINT - SEPT. '85
110 DATA 0,95,159,253,200,1,
49,181,199,999,179,247,0,999
120 DATA 156,253,200,1,49,18
1,999,160,253,200,1,49,0,999
,255,255,999
130 REM BY GEORGE F. STEFFAN
, LA 99ER COMPUTER GROUP, P
O BOX 3547, GARDENA CA 90247
140 REM BASED ON AN IDEA BY
PAUL YORKE : 1200 STARFISH L
ANE : STUART, FL  23494
150 REM DISK SYSTEM REQUIRED
160 REM OP$ = "95 OPEN #1:"
IN TOKENIZED STORAGE
170 REM EN$ = ",OUTPUT" IN C
ONDENSED DISK CODE (TOKENS)
180 REM E$=END OF PROGRAM
190 REM P$="PRINT #1:"
200 CALL CLEAR
210 PRINT " THIS PROGRAM WIL
L CONVERT  ANY NON-MODULE DE
PENDENT    PROGRAM TO PRINT
TO A NAMED OUTPUT DEVICE."
220 PRINT :" IT DOES THIS BY
 ADDING AN  OPEN STATEMENT A
ND REWRITING"
230 PRINT " ALL PRINT STATEM
ENTS ADDING OUTPUT REQUIREME
NTS."
240 PRINT :" PROGRAM MUST HA
VE ONLY ONE STATEMENT PER LI
NE."
250 PRINT " THE ORIGINAL PRO
GRAM MUST  BE SAVED IN MERGE
 FORMAT."
260 PRINT :: INPUT " PRESS E
NTER TO CONTINUE":T$
270 PRINT :" YOU MUST RESEQU
ENCE YOUR   PROGRAM BEFORE S
AVING IT IN MERGE FORMAT.";;
:
280 GOSUB 530 :: OP$=T$
290 GOSUB 530 :: EN$=T$
300 GOSUB 530 :: P$=T$
310 GOSUB 530 :: C$=T$
320 GOSUB 530 :: E$=T$
330 PRINT :: INPUT "PROGRAM
TO BE CONVERTED?    ":IF$
340 PRINT :: INPUT "NAME OF
MODIFIED PROGRAM?   ":OF$
350 IF OF$=IF$ THEN PRINT "N
AMES MUST BE DIFFERENT!" ::
GOTO 330
360 PRINT :: LINPUT "NAME OF
 OUTPUT DEVICE?      ":OD$
370 PRINT :"A - ADD OUTPUT T
O DEVICE";:"C - CHANGE FROM
 SCREEN TO   OUTPUT DEVICE":
;:"SELECTION"
380 ACCEPT AT(23,12)SIZE(-1)
VALIDATE("AC")BEEP:T$ :: S=-
5*(T$="A")
390 OPEN #1:IF$,DISPLAY ,VAR
IABLE 163,INPUT
400 OPEN #2:OF$,DISPLAY ,VAR
IABLE 163,OUTPUT
410 PRINT #2:OP$&CHR$(LEN(OD
$))&OD$&EN$ :: P=1
420 IF EOF(1)THEN GOTO 460 E
LSE LINPUT #1:T$ :: IF T$=E$
 THEN GOTO 460
430 GOSUB 570 :: IF C=156 TH
EN L2=L2+S :: GOSUB 560 :: P
RINT #2:LN$&P$&SEG$(T$,4,160
)
440 IF C=139 OR C=152 THEN G
OSUB 560 :: GOSUB 510 :: L2=
L2+5 :: GOSUB 560 :: PRINT #
2:LN$&SEG$(T$,3,161)
450 GOTO 420
460 L2=L2+10 :: GOSUB 560 ::
 GOSUB 510
470 PRINT #2:E$ :: CLOSE #1
:: CLOSE #2
480 PRINT :;:"TO GET YOUR PR
OGRAM YOU MUSTDO THE FOLLOWI
NG:":;:"NEW":;:"MERGE ";IF$:
;:"MERGE ";OF$
490 PRINT :"THE CHANGED PROG
RAM WILL    THEN BE IN MEMOR
Y AND YOU   SHOULD SAVE BEFO
RE RUNNING  IT."
500 STOP
510 IF P THEN PRINT #2:LN$&C
$ :: P=0
520 RETURN
530 T$="" ! CLEAR STRING
540 READ C :: IF C<256 THEN
T$=T$&CHR$(C) :: GOTO 540
550 RETURN
560 LN$=CHR$(L1-(L2>255))&CH
R$(L2+256*(L2>255)):: RETURN
570 L1=ASC(T$):: L2=ASC(SEG$
(T$,2,1)):: C=ASC(SEG$(T$,3,
1))::RETURN
```

ERROR TRAPPING TECHNIQUES - By Ted Mills, CALL SOUNDS Newsletter, Central Westchester 99'ers, May, 1986

(Editorial Remarks by Art Byers, C.W. 99'ers)

Computers generally have built-in error handling procedures. At a minimum a computer will stop when it encounters an error condition. But first the computer will store certain information, at designated memory addresses, concerning the type of error encountered and the line where the error occurred. On my Apple these error messages can only be accessed by PEEKing into memory through an error handling subroutine written into the program. Otherwise the program simply stops when an error occurs. The TI 99/4A, however, not only routinely describes the error type but the line where it was encountered as well. (In addition the 99/4A's TI BASIC has some built-in error routines that do not stop a program but rather issue a warning. One example is entering an alphabet value into an INPUT statement that expects a numerical value. Another: Extended BASIC's ACCEPT AT statement allows you to VALIDATE the type of data you want entered and will give you a WARNING "honk" and refuse to accept any other than the data specified. See page 48 of the XB manual - Ed.)

MS-DOS computers feature only a slight improvement in error handling in that the line is actually displayed after the program stops and places the cursor over the actual error.

Error handling functions are not only used to trap errors in newly written, or typed-in, programs, but also error handling routines have useful programming applications. The latter were the initial purpose of this article. However, some general comments might also be appropriate.

Extended BASIC has two error statements - ON ERROR and CALL ERR. ON ERROR simply tells the computer what to do when an error condition is encountered. Generally, ON ERROR will GOTO or GOSUB to a subroutine.

ON ERROR can be used in many ways. The most common is to keep programs from crashing when the user does something wrong such as trying to load a blank or not initialized data disk, hardware goofs, i.e., you left the door open on the disk drive, or you misspelled PIO as PI0.

CALL ERR is best used for debugging a program. Once the program is error free, the CALL ERR lines can be deleted. The Syntax of the CALL ERR subprogram contains four variables describing some aspect of the error condition. The statement is in the form CALL ERR(Error Code, error type, severity, line number). Error type simply distinguishes between program errors and input/output errors. Frankly, I never have understood the usefulness of the severity message. (Neither have I! - Ed.)

So far so good! If the error is in the line where the error condition was encountered, life becomes relatively simple. However, the error may originate somewhere else, such as a bad value generated earlier that does not show up until later. The best procedure, therefore, is to place an ON ERROR statement near the beginning of the program that GOSUBs or GOTOs an error trapping routine at the end. The subroutine should include a CALL ERR subprogram. Once the error codes and the line are identified then PRINT statements can be added to the subroutine to print out each of the variables in the line where the error condition was encountered. Watch out, though, for BAD VALUEs arising from an improper use of reserved words. I once typed in a program, written in TI BASIC, using Extended BASIC. The TI BASIC version had a variable DIGIT which is an Extended BASIC reserved word.

The TRACE command is a useful supplementary debugging tool. However, I prefer to insert "I'M HERE AT (LINE)" to follow program flow. If you do use TRACE, especially on a long and involved program, it is helpful to have a screen dump in low memory to print the TRACE flow on to paper. The one by Qualitysoft works very well. (Westchester also has one in the club library for free.)

The ON ERROR statement should be a useful programming tool. I routinely insert ON ERROR statements in my program that either return to the main menu if an error occurs or saves whatever data has been entered so far to disk. It is very exasperating to lose a lot of data when a program comes to a screeching halt due to an error. Similarly, ON ERROR can be used to close a file.

Last Fall I typed in a stock charting program that could chart a lot of price data that I had acccumulated. Among the inputs for each data point were the day, month and year. These I entered in through READ/DATA statements. To check for typing accuracy, and to count the weeks, I included a subroutine which read and printed the data items. Instead of using an end of data identifier I simply used an ON ERROR message to save the data to disk as soon as I had run out of DATA statements.

Some programmers hold forth that a fully debugged and properly written program should not need error traps, except to guard against the hardware errors discussed above. They consider use of ON ERROR as a programming tool to be somewhat inelegant, but I believe it provides an important measure of safety which I like.

One final comment. It is possible to have many ON ERROR routines in the same program, as long as each one is turned on and off at the right time. For example, I usually insert an "ON ERROR GOTO (Menu)". However, an "ON ERROR (Save File)" heads my insert data routine. After the file is saved then I return to the "ON ERROR GOTO (Menu)" command.

```
100 ! ***** MS/LABELS ***** By: Martin A. Smoley ***** For EPSON Printer *****
110 !                ***** NorthCoast 99er's UG *****
120 OPEN #9:"PIO" !  OPEN PRINTER (Could be RS232)    *** Extended Basic ***
130 PRINT #9:CHR$(27);"O";CHR$(27);"8";!
       "O"=STOP skip over perf,"8"=STOP paper end detector
140 CALL CLEAR :: CALL SCREEN(13)
150 PRINT "     ** MS/LABELS **": :"        PRINTS": :"    3-1/2in BY 15/16in":
 :"          LABELS": : :
160 PRINT " Enter Data at Prompts'": :" You will have 4 line per": :" label.  Li
ne #1 = 15 Cols.": :"          Line #2 = 28 Cols.": :
170 PRINT " Lines #3 and #4 = 49 Cols.": : :
180 GOSUB 190 :: GOSUB 210 :: GOSUB 220 :: GOSUB 230 :: GOTO 240
190 PRINT :: PRINT "              ^^^^^^^^^^^^^^^"
200 INPUT "ENTER LINE 1 ":A$ :: RETURN
210 PRINT :: PRINT "  ENTER LINE #2" :: INPUT "^^^^^^^^^^^^^^^^^^^^^^^^^^^^":B$
:: RETURN
220 PRINT :: PRINT "  ENTER LINE #3" :: INPUT "0^^^^^^^^^1^^^^^^^^^2^^^^^^^^^3^^^
^^^^^^4^^^^^^^49        ":C$ :: RETURN
230 PRINT :: PRINT "  ENTER LINE #4" :: INPUT "0^^^^^^^^^1^^^^^^^^^2^^^^^^^^^3^^^
^^^^^^4^^^^^^^49        ":D$ :: RETURN
240 PRINT :: INPUT "HOW MANY COPYS ":X
250 CALL CLEAR :: PRINT "  Hold >Q< to Quit Printing": : : : : :
260 FOR I=1 TO X !    ******** PRINTOUT LOOP ********
270 ! PRINT #9:CHR$(27);"G";!                START DOUBLE STRIKE OPTIONAL
280 PRINT #9:CHR$(27);"E";!                   START EMPHASIZED
290 ! PRINT #9:CHR$(27);"M";!               Start Elite-size(makes #1=18 characters)
300 PRINT #9:CHR$(27);"W";CHR$(1);!            START ENLARGED
310 PRINT #9:A$
320 PRINT #9:CHR$(27);"W";CHR$(0);!            STOP ENLARGED
330 ! PRINT #9:CHR$(27);"P";!               Stop Elite-size(Needed if 290 is used)
340 PRINT #9:" ";B$;CHR$(27);"F" !            STOP EMPHASIZED
350 PRINT #9:CHR$(27);CHR$(15);" ";C$;:;" ";D$;CHR$(18);CHR$(27);"H";!
CHR$(15)=START CONDENSED+CHR$(18)=STOP,"H"=STOP DOUBLE STRK.
360 FOR K=1 TO 3 :: PRINT #9 :: NEXT K
370 CALL KEY(0,K,S):: IF K=81 OR K=113 THEN 390
380 NEXT I
390 CALL CLEAR :: CALL SCREEN(6)!      ******** Beginning of TASK SCREEN ********
400 PRINT " Enter M for More labels": :"       N for New labels": :"       L to
Change a line": :
410 PRINT "        Q to Quit the program": :
420 INPUT " Enter your chioce: ":DO$
430 IF DO$="M" OR DO$="m" THEN CALL CLEAR :: GOTO 240
440 IF DO$="N" OR DO$="n" THEN 140
450 IF DO$="L" OR DO$="l" THEN 480
460 IF DO$="Q" OR DO$="q" THEN 520
470 GOTO 420
480 CALL CLEAR !               ******** Beginning of LINE CHANGE SCREEN ********
490 INPUT "  Enter line number to be          changed  1 to 4 ":L :: IF L<1 OR
L>4 THEN 490
500 ON L GOSUB 190,210,220,230
510 GOTO 390
520 PRINT #9:CHR$(27);"@";! Initialize Printer = Wipe out any leftover commands
530 CLOSE #9
540   *** MS/LABELS ***
550 END
```

MS/LABELS-DOC

"MS/LABELS" started out to be a small, simple program to print 3-1/2 in X 15/16 in. labels for return addresses and disk labels, but it evolved into the program you see at the left.

THE USER INSTRUCTIONS FOLLOW

(1) Load the program (Don't run it yet).

(2) Align your labels in the printer then turn the printer on.

(3) Now RUN the program.

(4) Enter the data as prompted by the program. There is one circumflex (^) for each space on the entry line. Do not use any commas.

(5) After you have entered (4) lines the program will ask how many labels you want. If you want to see one enter 1. After the label is printed you will see a screen which will let you print (M)ore if you like what you see.

(6) If you don't like them enter L to change a line and then the line number you would like changed. You can repeat the L for as many lines as you need, or you can use M for more and print one at any time until you like the label you have. At this point you use More, then type in the quantity you want and the printer will start running them off.
If you change your mind, HOLD >Q< until the printer stops and you will return to the task screen.

(7) At the task screen you can also enter an (N) if you want a completely New label or (Q)uit to exit the program.

NOTE: If your ribbon is not dark enough you can edit the program and delete the (!) and the space from the beginning of line 270 This will give you Double Strike throughout. Also! Doing the same thing to line Nos. 290 and 330 will give you 18 characters in line #1 if your printer is capable of Elite Print (You will have to remember that you have (3) characters past the last (^) in line one.)

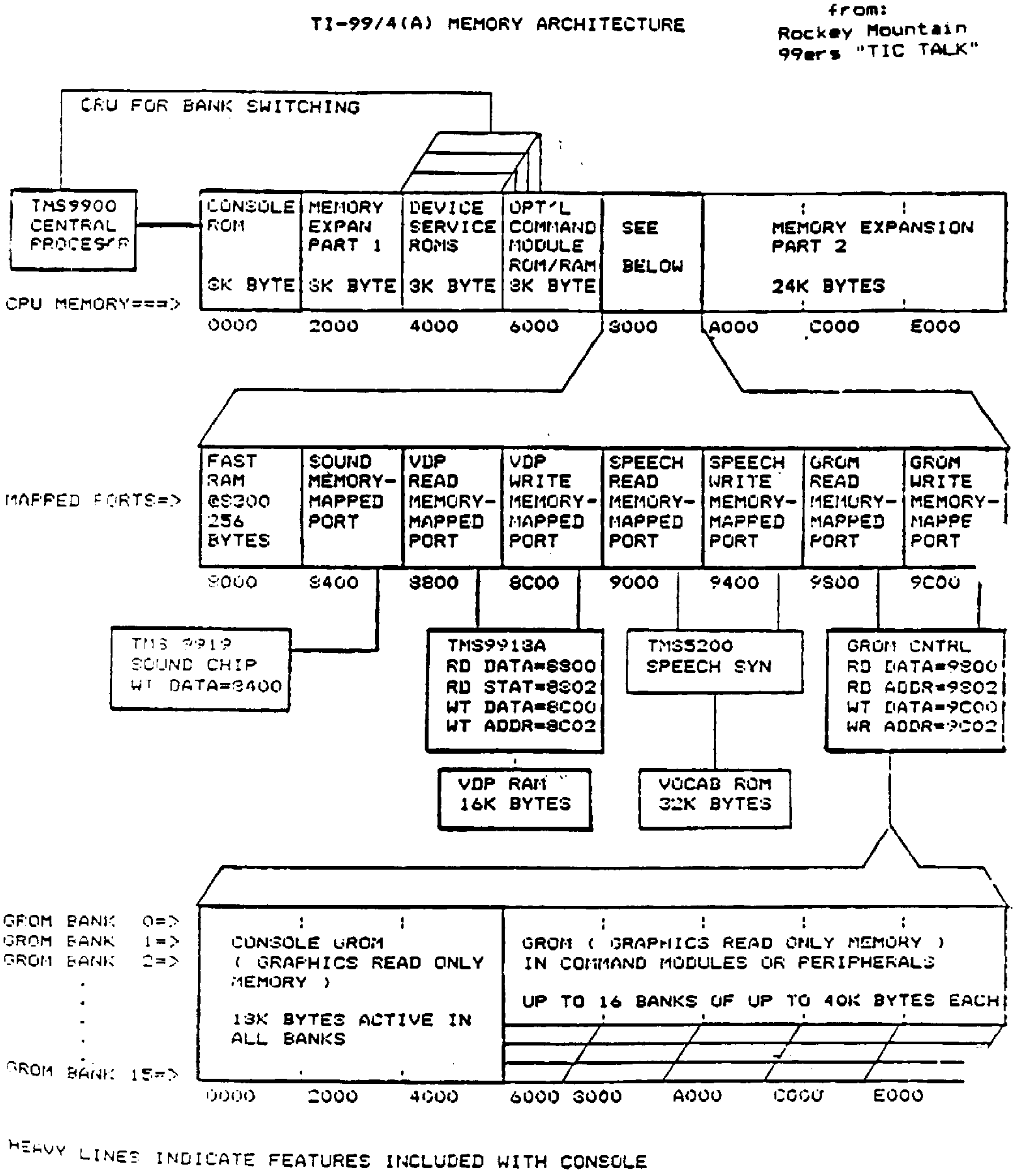
If you do not like to type, my programs are in the NorthCoast 99er's Library.
Good Luck! Marty

-- MS/LABELS --
TI99/4A   Extended Basic
This label was made by the program listed above.
Ln.#1=ENLARGED   #2=Std. size   #3&#4=Condensed

Study Brings Benefits . . .

Squeezing Real Benefits From Your 99/4A and Geneve Systems . . . Assembly Language

This block diagram came from TI and may be of general interest-
a picture is worth a thousand words!


TI-99/4(A) MEMORY ARCHITECTURE
from:
Rockey Mountain
99ers "TIC TALK"
CRU FOR BANK SWITCHING
TMS9900 CENTRAL PROCESSOR
CPU MEMORY===>
CONSOLE ROM 8K BYTE
MEMORY EXPAN PART 1 8K BYTE
DEVICE SERVICE ROMS 8K BYTE
OPT'L COMMAND MODULE ROM/RAM 8K BYTE
SEE BELOW
MEMORY EXPANSION PART 2 24K BYTES
0000 2000 4000 6000 8000 A000 C000 E000
MAPPED PORTS=>
FAST RAM @8300 256 BYTES
SOUND MEMORY-MAPPED PORT
VDP READ MEMORY-MAPPED PORT
VDP WRITE MEMORY-MAPPED PORT
SPEECH READ MEMORY-MAPPED PORT
SPEECH WRITE MEMORY-MAPPED PORT
GROM READ MEMORY-MAPPED PORT
GROM WRITE MEMORY-MAPPE PORT
8000 8400 8800 8C00 9000 9400 9800 9C00
TMS 9919 SOUND CHIP WT DATA=8400
TMS9918A RD DATA=8800 RD STAT=8802 WT DATA=8C00 WT ADDR=8C02
TMS5200 SPEECH SYN
GROM CNTRL RD DATA=9800 RD ADDR=9802 WT DATA=9C00 WR ADDR=9C02
VDP RAM 16K BYTES
VOCAB ROM 32K BYTES
GROM BANK 0=>
GROM BANK 1=>
GROM BANK 2=>
GROM BANK 15=>
CONSOLE GROM (GRAPHICS READ ONLY MEMORY)
18K BYTES ACTIVE IN ALL BANKS
GROM (GRAPHICS READ ONLY MEMORY) IN COMMAND MODULES OR PERIPHERALS
UP TO 16 BANKS OF UP TO 40K BYTES EACH
0000 2000 4000 6000 8000 A000 C000 E000
HEAVY LINES INDICATE FEATURES INCLUDED WITH CONSOLE

THE SCREEN PAGER UTILITY
By Michael St. Vincent

How often have you wanted to look at part of a program as it runs or set up an initial instruction screen that could be stored and recalled in an instant? If you are familiar with the almost complete impossibility of doing this, especially in the Extended BASIC environment and want to get free of such limits, here is your answer: an assembly language subroutine that is short and non-complex.

Simple solutions to problems such as screen storage are often overloked in favor of staying strictly in one language's environment. Most people are unfamiliar with the usefulness of having machine language routines take over chores that are much slower in BASIC. To store a screen in BASIC, for example, most programmers would use a GCHAR to read all of the screen and store the result in an array. Besides being slow and inefficient, a BASIC routine to do such would use large amounts of memory.

Enter the amazing and fast 9900 machine language routine! The screen, usually a set of rows and columns to a BASIC programmer, becomes only a set of memory locations. In this form, moving a copy of the screen becomes as simple as assigning the assembly equivalent of a few variables and a GOSUB. Operation of the subroutines is kept simple by having the computer do the calculating. The possible applications of these subprograms are limited only to the programer's imagination.

How the program is used;

The subroutines, once assembled, are some of the simplest to use. Loading the programs into memory is accomplished by using a CALL INIT command followed by a CALL LOAD("DSK1.PAGER/OBJ") command. The routines are automatically stored in the memory and become invisible until needed. Four programs are loaded simultaneously for use in Extended BASIC: PGSAV1, PGSAV2, PGSHO1, AND PGSHO2. The SAV programs save everything on the screen at the instant they are called to pages 1 and 2 respectively. The SHO programs return the previously saved pages to the screen. All four programs are accessed by CALL LINK("pgname") where pgname is one of the program names given above. The amount of time spent by the programs can only be measured in microseconds. Using OLD, SAVE, MERGE, and NEW commands have no effect on the screens stored in memory (thus, one could list a program, save a screen of the list, load a new program, and still be able to look at the listing of the old program). The only restrictions on the programs are that they only store the characters, neither the colors nor any sprites are kept.

How the program works:

The programs in assembly use a simple system of setting up a block of CPU RAM to store pages. Once a screen is to be stored, the registers 0, 1, and 2 are loaded with the address of the screen map in VDP RAM (000), the address of the CPU RAM block, and the number of bytes to transfer (768 for the full screen). A simple BLWP (branch and link with workspace pointer) command links to another utility routine which does the actual transfer. After the transfer is completed, the program uses the psuedo-opcode RT to reset the workspace pointer to the BASIC nterpeter area from where it branched. At that point, the BASIC level ogram continues to execute.

How to assemble and install this program on your disks:

Using the Editor/Assembler package, type in the source listing which follows exactly as shown. Spacing is important to insure that the program will assemble properly. Once the program is typed in (you do need to copy the remarks that are preceeded by an asterisk, store t source code (what you typed) under the filename "PAGER/SOU". Then l the Assembler. When asked for the source filename, gi "DSK1.PAGER/SOU", and when asked for the object filename, give "DSK1.PAGER/OBJ". If you have a printer, give the device name at the prompt, otherwise, hit <enter>. The options for assembly are "RSL" if you have given a printer device name, of "RS" if you haven't. The assembler should do its job within 5 minutes and should print "0000 ERRORS" at the end. If there are any errors during assembly, refer to the source listing of this newsletter and compare what you typed.

As listed, the program assembles with no errors.

```
* THE SCREEN PAGER UTILITY
* SOURCE CODE WRITTEN BY MICHAEL ST. VINCENT
* USED TO STORE UP TO 2 SCREEN-FULLS FOR LATER USE
*
       DEF   PGSAV1,PGSAV2,PGSHO1,PGSHO2  * NAME ROUTINES
*
VMBW   EQU   >2024         * VDP WRITE ROUTINE
VMBR   EQU   >202C         * VDP READ ROUTINE
SCRMAP EQU   >0000         * START OF SCREEN MAP ADDRESS
SCRCNT EQU   768           * NUMBER OF CHARACTERS IN MAP
*
PAGE1  BSS   768           * STORAGE BUFFER 1
PAGE2  BSS   768           * STORAGE BUFFER 2
*
PGSAV1 LI    R1,PAGE1      * ACTIVATE BUFFER 1
       JMP   GOSAVE        * GOTO THE SAVE ROUTINE
PGSAV2 LI    R1,PAGE2      * ACTIVATE BUFFER 2
GOSAVE LI    R0,SCRMAP     * STARTING POINT TO READ FROM MAP
       LI    R2,SCRCNT     * NUMBER OF BYTES TO MOVE
       BLWP  @VMBR         * "GOSUB" TO READ
       RT                  * RETURN TO BASIC
*
PGSHO1 LI    R1,PAGE1      * ACTIVATE BUFFER 1
       JMP   GOSHOW        * GOTO THE RESTORE ROUTINE
PGSHO2 LI    R1,PAGE2      * ACTIVATE BUFFER 2
GOSHOW LI    R0,SCRMAP     * STARTING POINT TO REPLACE MAP
       LI    R2,SCRCNT     * NUMBER OF BYTES TO MOVE
       BLWP  @VMBW         * "GOSUB" TO WRITE BACK TO MAP
       RT                  * RETURN TO BASIC
*
       END                 * TELL ASSEMBLER TO STOP
```

===

If you want to use this program in BASIC with the Editor/Assembler module, change the lines to match this header:

```
*
       DEF   PGSAV1,PGSAV2,PGSHO1,PGSHO2  * NAME ROUTINES
       REF   VMBR,VMBW
*
SCRMAP EQU   >0000         * START OF SCREEN MAP ADDRESS
SCRCNT EQU   768           * NUMBER OF CHARACTERS IN MAP
*
```

Operation of the program is the same as desribed for Extended BASIC.

CALL PEEK

Hello again everyone. There's a lot to cover this time, so let's get right to it. Last month's A/L Challenge was to write a program to input a line from the keyboard and output it to a printer. Since nobody called to ask a question about device I/O, I assume everybody was able to get all the information they needed from the materials they have. Everybody DID write a program didn't they??? Just in case, I'll cover a few high points before presenting my solution to the challenge. Most of the needed information, although a bit cryptic, can be found in the Editor/Assembler manual. Due to space requirements (and laziness on my part), I'll not reprint that information here, but will offer a few comments on it. So... grab the manual and let's look at "File Management".

As I said last month, one of the great things about our computer is the ability for our programs to interface with most peripherals in the same manner regardless of the type device. This is due to the use of "smart" peripheral controllers and the "file" concept. Read pages 291 and 292 in the E/A manual for a description on the "file" concept. Any device, with the exception of the cassette recorder (see the note on page 262), that can be accessed with the OPEN, CLOSE, INPUT and/or PRINT statements in basic can be accessed in assembly language, using a common subroutine provided in the E/A utilities called DSRLNK. Each peripheral card contains a DSR (Device Service Routine) that handles the I/O to that device and makes data flow to and from the device appear to us as a "file". The DSRLNK subroutine takes care of locating the desired device, ie. "PIO" or "DSK", and interfacing it with our program. Page 262 of the E/A manual contains a description of how to use DSRLNK.

The key to accessing any device with the DSRLNK utility is the PAB (Peripheral Access Block). The PAB is a group of data that defines all information necessary to access a particular file on whatever device we are working with. The PAB has a strict format, and is always located in VDP RAM. Pages 293 and 294 in the E/A manual cover the format of a PAB. A PAB is 10 bytes long, plus the length of the file descriptor. A file descriptor is the device name, file name, and any options needed for a particular device. "DSK1.MYFILE" and "RS232.BA=2400" are examples of a file descriptor. The E/A manual has a pretty good description of the PAB, but here are a few good things to remember... The PAB is a two way street. In addition to its function of passing necessary information to the device, the device also uses the PAB to pass necessary information back to our program. For example, byte 1 of the PAB is used by the device to identify any errors encountered during the current operation and byte 5 is used by the device to tell us the number of bytes read during a READ operation. Remember that byte 1 is a bit mapped byte, that is, more than one piece of information is passed through this one byte. Also, bytes 2-3 and 6-7 are taken as word (16 bit) values. The data buffer address in bytes 2 and 3 always point to a buffer area in VDP RAM. This is where you put data that will be written to a file before linking to the device, and it is where data read from a file will be placed by the device. It is important to remember that you can change the data buffer address between each link to the device if necessary. For instance, you could use separate read and write buffers when dealing with relative files. However, you must be careful to place the data buffer in a VDP RAM location that will not interfere with the operation of the computer. Addresses between >1000 and >3000 are usually a good choice for PABs and data buffers in a program running out of the E/A module.

Pages 295 thru 298 of the E/A manual describe the meanings of the I/O opcodes used in byte 0 of the PAB. Page 299 describes possible error conditions. Although it is good to know how error codes are passed back from a device, the DSRLNK routine transfers the error code to register 0 of the calling workspace, and sets the equal bit in the status register if an error occurs during access to a device. See page 262 for more information. Hopefully, these few comments will answer any questions you may have had. If not, feel free to call. Now... here's my solution to A/L Challenge #2.

```
            TITL 'A/L CHALLENGE #2'
            REF  DSRLNK,VSBW,VMBW,KSCAN,VDPWD,GRMRA,GRMWA
            DEF  START
     PAB    EQU  >1000          Location of PAB
     BUFFER EQU  >38                   .    .  data buffer
     NAMLEN EQU  PAB+9                 .    .  name length in PAB
     COUNT  EQU  PAB+5                 .    .  data count in PAB
```

```
PNTR   EQU  >8356            .    .  name length pointer
KEY    EQU  >8375           Keyscan returns character here
STATUS EQU  >837C           Location of GPL status byte '
GPLWS  EQU  >83E0             .    .  GPL registers
MAX    EQU  80              Max char count
WKSP   EQU  >8300           Use this area for registers
*
OPEN   BYTE 0               I/O opcode for OPEN
CLOSE  BYTE 1                .    .     .  CLOSE
WRITE  BYTE 3                .    .     .  WRITE
BREAK  BYTE 2               Keycode for FCTN 4
ENTER  BYTE >D                .    .  ENTER
CURSOR BYTE >1F             Char number of cursor
*
* Data for PAB
       EVEN                 Force even address
PABDAT BYTE 0               I/O opcode
       BYTE >12             File type description
       DATA BUFFER          Data buffer address
       BYTE MAX             Record length
       BYTE 0               Character count
       DATA 0               Record number
       BYTE 0               Screen offset
       BYTE ENDAT-FILNAM Name length
FILNAM TEXT 'PIO'           Filename
ENDAT  EQU  $               Mark end of data
*
       EVEN                 Force even address
GRMSAV BSS  2               Save GROM address here
* Save GROM address fisrt
START  MOVB @GRMRA,@GRMSAV   Get MSB
       NOP                   Waste time
       MOVB @GRMRA,@GRMSAV+1 Get LSB
       DEC  @GRMSAV          Adjust
*
       LWPI WKSP            Load the workspace pointer
       LI   R0,PAB          R0 points to PAB
       LI   R1,PABDAT       R1 points to data for PAB
       LI   R2,ENDAT-PABDAT R2 has byte count
       BLWP @VMBW           Write the data to VDP
       MOVB @OPEN,R1        OPEN opcode to R1
       BL   @IO             Open the file
*
LOOP1  BL   @CLS            Clear the screen
       LI   R0,BUFFER       R0 points to screen location
       CLR  R3              Use R3 for character count
LOOP2  MOVB @CURSOR,R1      R1 has cursor char
       BLWP @VSBW           Put cursor on screen
LOOP3  BLWP @KSCAN          Get a keypress
       MOVB @STATUS,@STATUS Check for new keypress
       JEQ  LOOP3           Loop if no new key
       MOVB @KEY,R1         Key code to R1
       CB   R1,@BREAK       FCTN 4?
       JEQ  ENDIT           Yes, prepare to end
       CB   R1,@ENTER       ENTER key pressed?
       JEQ  PRINT           Yes, write line to device
* Just an ordinary character, put it on screen
       BLWP @VSBW           Write to screen
       INC  R3              Increment counter
       INC  R0              Increment screen pointer
```

```
        CI   R3,MAX        Reached max line length?
        JLE  LOOP2         No, continue
* Maximum line length reached if here
        DEC  R3            Adjust count
        DEC  R0            and screen pointer
        JMP  LOOP2         Get another key
*
PRINT   MOV  R3,R1         Char count to R1
        SWPB R1            and into left byte
        LI   R0,COUNT      Pointer to count byte in PAB
        BLWP @VSBW         Write the count byte
        MOVB @WRITE,R1     WRITE opcode to R1
        BL   @IO           Output the line
        JMP  LOOP1         Get another line
*
ENDIT   MOVB @CLOSE,R1     CLOSE opcode to R1
        BL   @IO           Close the file
* Restore GROM address
        MOVB @GRMSAV,@GRMWA   Write MSB
        NOP                   Waste time
        MOVB @GRMSAV+1,@GRMWA Write LSB
*
        LWPI GPLWS         Load GPL registers
        B    @>6A          Return to E/A module
*
* CLS Subroutine, clears the screen
* Uses R0, R1, and R2
*
CLS     CLR  R0            Beginnig screen location
        LI   R1,>2000      Space char in left byte, R1
        BLWP @VSBW         Clear first byte
        LI   R2,767        Remainder count
CLSLP   MOVB R1,@VDPWD     Clear next byte
        DEC  R2            Decrement count
        JNE  CLSLP         Loop til done
        RT                 then return
*
* I/O Subroutine
* Enter with I/O opcode in R1 (left byte)
* Handles errors, ignores CLOSE errors
* Uses R0, R1, R2, and R3
*
IO      LI   R0,PAB        Point to PAB
        BLWP @VSBW         Write the I/O opcode
        LI   R0,NAMLEN     Point to name length in PAB
        MOV  R0,@PNTR      Required for DSRLNK
        BLWP @DSRLNK       Link to device
        DATA 8             For file I/O
        JEQ  ERROR         Handle any errors
IORET   RT                 Return if no errors
*
ERROR   CB   @CLOSE,R1     Closing the file?
        JEQ  IORET         Yes, ignore error
        SRL  R0,8          Move error code to right byte
        SLA  R0,1          and multiply by 2
        MOV  R0,R3         Save error code in R3
        BL   @CLS          Clear the screen
        LI   R0,BUFFER+2   Location for error msg
        MOV  @ERRTAB(R3),R1 Message location to R1
        MOVB *R1+,R2       Byte count to R2
```

```
0128          SRL  R2,8          Adjust to word value
0129          BLWP @VMBW         Write the message to screen
0130 *
0131          LI   R0,742        Screen location
0132          LI   R1,PRESS      Message pointer
0133          LI   R2,21         Byte count
0134          BLWP @VMBW         Write to screen
0135 *
0136 ERRLP    BLWP @KSCAN        Get a key
0137          MOVB @STATUS,@STATUS New key?
0138          JEQ  ERRLP         Not yet
0139          CB   @ENTER,@KEY   ENTER?
0140          JNE  ERRLP         No, try again
0141          BL   @CLS          Clear the screen
0142          B    @ENDIT        End the program
0143 *
0144 ERRTAB   DATA BDNMSG        Bad device name
0145          DATA DWPMSG        Device write protected
0146          DATA BOAMSG        Bad open attribute
0147          DATA ILOMSG        Illegal operation
0148          DATA OBSMSG        Out of buffer space
0149          DATA EOFMSG        End of file
0150          DATA DVCMSG        Device error
0151          DATA FILMSG        File error
0152 *
0153 BDNMSG   BYTE 16
0154          TEXT 'Bad Device Name!'
0155 DWPMSG   BYTE 26
0156          TEXT 'Device is write protected!'
0157 BOAMSG   BYTE 19
0158          TEXT 'Bad OPEN attribute!'
0159 ILOMSG   BYTE 18
0160          TEXT 'Illegal operation!'
0161 OBSMSG   BYTE 20
0162          TEXT 'Out of Buffer Space!'
0163 EOFMSG   BYTE 25
0164          TEXT 'Attempt to Read Past EOF!'
0165 DVCMSG   BYTE 13
0166          TEXT 'Device Error!'
0167 FILMSG   BYTE 11
0168          TEXT 'File Error!'
0169 PRESS    TEXT 'Press <ENTER> to end.'
0170          END
```

Here's a short description of how the program works...

Lines 1-3 assign a title to the assembly listing, inform the assembler which pre-defined utilities and symbols we'll be using, and defines START as a label to be placed in the ref/def table when the program is loaded.

Lines 4-13 equate various labels to values to be used in the program. Notice that the value field for the labels NAMELEN and COUNT contain "well defined expressions". These labels are referenced from the label PAB. By using expressions such as these you are able to change the values of several related labels by changing only one line in the program. See page 49 in the E/A manual for the description of a well defined expression.

Lines 15-20 place 6 one byte values in the object code that will be used by various routines in the program.

Line 23 contains an EVEN directive. This directive tells the assembler to make sure the location pointer is at an even address. Although at this point the location pointer would be at an even address, that could change if you added another byte value before line 22. It's good practice to add an EVEN directive after using one or more BYTE or TEXT directives in your program. The reason for needing to be sure that we're at an even address at this point in the program is due to the use of the DATA directive in line 26. Remember that the BYTE directive places one byte of data in the program while DATA places one word or two bytes in the program. When the assembler encounters the DATA directive in the source code, it will increment the location pointer to an even address if it should happen to be at an odd location. So... if at line 24 the location pointer were at an odd address the result would be a one byte 'hole' in the object code between lines 25 and 26. This would result in a PAB that would not conform to the strict format that must be followed.

Lines 24-33 contain data that will make up the PAB. Notice that another well defined expression is used for the name length in line 31. Doing this allows you to change the filename in line 32 without having to change the filename length byte, as long as you do not place anything between the end of the filename and the label ENDAT in line 33. If your printer is connected to the system through a device other than PIO, you'll have to change the filename. ENDAT is equated to the current location pointer through the use of the dollar sign. The assembler recognizes the dollar sign to mean the current value of the location pointer. Actually, ENDAT and GRMSAV have the same value so we could have used GRMSAV in line 31 and done away with the label ENDAT. However, I think it's a good idea to keep related sections of code together. If GRMSAV had been used in place of ENDAT, the name length byte would get screwed up if any code was added before GRMSAV while writing the program. You'll notice that I've used empty comment lines to keep the code in modular form.

Line 35 contains another EVEN directive to ensure that the following code begins on an even address, regardless of the length of the PAB data.

Line 36 reserves 2 bytes in the object code to be used to save the GROM address pointer.

Line 38 is where the program will start to execute. Lines 38-41 save the GROM address. This must be done because some devices alter the GROM address when accessed. In order to return to the E/A module the GROM address must be the same when we leave our program as it was when the program was entered. The code for saving the GROM address came directly from pages 270 and 271 in the E/A manual.

Line 43 sets the workspace register to >8300.

Lines 44-49 set up the PAB in VDP RAM and open the file. Again, an expression is used in line 46 to calculate the number of bytes contained in the PAB data.

Lines 51-81 comprise the main program loop. Actually this section is made up of three nested loops. The inner loop (lines 56-58) scans the keyboard for a new kwypress. The middle loop (lines 54-69) evaluates the keypress and takes necessary action depending on what key was pressed. It also places the cursor on the screen. This loop is executed once for each new keypress. Lines 66-73 keep up with the character count and make sure the 80 character limit is not exceeded. If the 80th character is entered, control is passed directly to the inner loop after the character is displayed on the screen. This prevents the cursor from overwriting the last character. The main, or outer loop, clears the screen and character counter and sets R0 to the starting screen location. You have probably noticed that I have chosen to use the area of VDP RAM that represents screen data for the output buffer. Since the data that we're writing to the printer is already stored on the screen, there's no need to move it to another area of VDP RAM before sending it to the printer. You can also use the screen data area as an input buffer when reading data if you need to display the data after reading it. The main loop also contains the routine used to send data to the printer. Lines 75-81 take care of this chore by placing the character count in the PAB, indicating a WRITE operation, and calling the subroutine IO to actually access the printer.

Lines 83-91 are executed when the break (FCTN 4) key is pressed. This routine closes the file, restores the GROM address, and returns control to the GPL interpreter after setting the workspace pointer to the GPL register area. This method of returning from the program is a modified version of the one on page 442 in the E/A manual. The manual suggests to clear the GPL status byte and then branch to location >0070. I prefer to branch to location >006A since the code there clears the GPL status byte. This saves a little memory usage in your program. Beginning on page 440 of the E/A manual are descriptions of several ways of returning to the system when your program ends.

Lines 96-103 are a subroutine to clear the screen. This subroutine uses the E/A provided

VSBW routine to clear the first byte of screen memory and then accesses the VDP chip directly to clear the rest of the screen. I used the VSBW routine as an easy way to set up the VDP Write Address register.

Lines 110-169 make up the I/O subroutine. This subroutine assumes that the PAB is alre set up with the exception of the I/O opcode. The I/O opcode must be passed to the subroutine in the MSB of R1. The I/O opcode is written to the PAB, the >8356 pointer is set up to satisfy the requirements of the DSRLNK routine, and then the device is accessed via DSRLNK. If access is successful, the subroutine returns to the calling program. If an error occurs, an error message is printed and the program returns to the E/A module after you press the ENTER key. If an error occurs during a CLOSE operation, it is ignored. For an error during any other operation, the error code is transfered to the right byte of R0 and then multiplied by 2. The multiplication is accomplished by shifting the value left by one bit. The resulting value is stored in R3, the screen is cleared and R0 is loaded with the screen address for the error message. In line 126 the indexed addressing mode is used to load R1 with the address of the correct error message to be printed. Since each address in the table at ERRTAB is 2 bytes long it is necessary to multiply the original error code by 2. This was done in line 122. The address of ERRTAB plus the value in R3 is loaded into R1. Now R1 will point to the length byte preceeding the error message. This length is transfered into the left byte of R2 via workspace register indirect auto-incermenting addressing, and then R2 is made into a word value with the shift instruction in line 128. The result of all this is that R0 has the screen address, R1 points to the message to print, and R2 contains a byte count of the message. The VMBW routine is used to print the message on the screen. After the error message is printed, the "Press ENTER to end." message is printed on the last screen line and the program waits for the enter key to be pressed. After the enter key is detected control passes to the code at ENDIT where the file is closed and the program returns to the E/A module. Lines 144-151 are a table of addresses pointing to the error messages. All the entries in this table could have been entered on one line in the source code, I put them on separate lines so it would look more like a table. The error messages that follow are taken more or less from the error code meanings listed on page 299 of the E/A manual.

Line 170 contains the END directive that tells the assembler that it has reached the end of the source code.

Well, there you have it. Remember that it is not necessary for your version of the program to operate in exactly the same manner as mine. If it works, it's OK.

The A/L Challenge for next month sort of expands on what we've learned this month. Write a program that will allow you to enter letters from the keyboard onto the screen at any location. In other words, the keyscan routine will have to recognize the arrow keys in order to move the cursor around on the screen. You should also try for a blinking cursor and repeating keys. The keyscan routine should also check for FCTN 3 to clear the screen, FCTN 4 to end the program and FCTN 6 to save the entire screen to a specified device. When FCTN 6 is pressed, save a couple of screen lines to a buffer, clear them, and prompt for an output device. After the output device is specified, restore the prompt lines and output the entire screen to the device. After the screen is output, return to the keyboard input routine with the screen still intact. Since this program will allow you to save a screen to disk, let's also include a routine to recall a screen. Start the program off with a menu to select 'design a screen' or 'recall a screen'. Your screen design keyscan routine should return to this menu when FCTN 4 is pressed. The format use to save the data is up to you. If you have questions, feel free to call 764-7881 after 6 PM. Out of town folks can write Rt. 9, Box 460, Florence, AL 35630. Please include an SASE for reply.

Until next time....

Danny Michael

Assembly Routine Restart after QUIT
by Joseph H. Spiegel

There are several Extended BASIC programs now that use assembly language routines. The loader for these routines is quite slow. For that reason, it is somewhat annoying if you have to leave Extended BASIC for some reason, then return and wait for your routines to reload. The worst part is that, in many cases, the program still resides untouched in expansion memory. What has happened is that the low memory has had to be reinitialized and the REF/DEF table cannot be found. The following program will read a current REF/DEF table and create the proper CALL LOAD's to restore it if you must leave Extended Basic. It will also perform minimal checking to see if the program you want is still intact.

The program is used as follows (assuming you have saved the program on dis

(1) From Command mode, do a CALL INIT :: CALL LOAD("DSK1.object file")
(2) Type RUN "DSK1.REFRESTORE" (or whatever you saved the program as)
(3) Answer the prompt with the complete filename that you you wish the merged file to be saved as.
(4) The program will recreate the REF/DEF table in merged form and print the program names as it goes.
(5) You will be prompted to enter the program name for checking upon reload. Enter one of the names from the program list. Depending upon the location of the program in memory, a check of the program may be included in he merge file. This check consists of a comparison of four bytes at the start of the chosen program. If the four bytes are OK, the variable FLAG will be set to 1, otherwise ti will be 0. If the REFRESTORE program has overwritten the object file, you will be given the location of the entry point, and the program will complete the merge file without the check.
(6) After completion, the merge file may be merged into your Extended BASIC object loader program

The program is below:

```
5 !by J.  H.  Spiegel 6/85 TI6240
10 PRINT :: INPUT "MERGE OUTPUT FILE NAME? ":OUTFILE$
20 OPEN #1:OUTFILE$,DISPLAY,VARIABLE 163
30 CALL PEEK(8194,A,B,C,D)
40 PRINT #1:CHR$(0)&CHR$(1)&CHR$(157)&CHR$(200)&CHR$(4)&"INIT"&
CHR$(130)&CHR$(157)&CHR$(200)&CHR$(5)&"CLEAR"&CHR$(0)
50 PRINT #1:CHR$(0)&CHR$(2)&CHR$(157)&CHR$(200)&CHR$(4)&
"LOAD"&CHR$(183)&CHR$(200)&CHR$(4)&"8194"&CHR$(179);
60 PRINT #1:CHR$(200)&CHR$(1-(A>9)-(A>99))&STR$(A)&CHR$(179);
70 PRINT #1:CHR$(200)&CHR$(1-(B>9)-(B>99))&STR$(B)&CHR$(179);
80 PRINT #1:CHR$(200)&CHR$(1-(C>9)-(C>99))&STR$(C)&CHR$(179);
90 PRINT #1:CHR$(200)&CHR$(1-(D>9)-(D>99))&STR$(D)&CHR$(182)&CHR$(0)
100 E=256*C+D :: LN=3
110 FOR X=E TO 16382 STEP 8
120 CALL PEEK(X,F,G,H,I,J,K,L,M)
130 PRG$=CHR$(F)&CHR$(G)&CHR$(H)&CHR$(I)&CHR$(J)&CHR$(K):: PRINT PRG$,
140 PRINT #1:CHR$(0)&CHR$(LN)&CHR$(157)&CHR$(200)&CHR$(4)&"LOAD"&
```

```
CHR$(183)&CHR$(200)&CHR$(5)&STR$(X)&CHR$(179);
150 PRINT #1:CHR$(200)&CHR$(1-(F>9)-(F>99))&STR$(F)&CHR$(179)
&CHR$(200)&CHR$(1-(G>9)-(G>99))&STR$(G)&CHR$(179);
160 PRINT #1:CHR$(200)&CHR$(1-(H>9)-(H>99))&STR$(H)&CHR$(179)&
CHR$(200)&CHR$(1-(I>9)-(I>99))&STR$(I)&CHR$(179);
170 PRINT #1:CHR$(200)&CHR$(1-(J>9)-(J>99))&STR$(J)&CHR$(179)&
CHR$(200)&CHR$(1-(K>9)-(K>99))&STR$(K)&CHR$(179);
180 PRINT #1:CHR$(200)&CHR$(1-(L>9)-(L>99))&STR$(L)&CHR$(179)&
CHR$(200)&CHR$(1-(M>9)-(M>99))&STR$(M)&CHR$(182)&CHR$(0)
190 LN=LN+1 :: NEXT X
200 INPUT "PROGRAM TO BE CHECKED UPON STARTUP? ":CK$ ::
 CK$=CK$&RPT$(" ",6-LEN(CK$)):: Y=E
210 IF Y>16383 THEN PRINT "THAT PROGRAM NOT FOUND" :: GOTO 200
220 CALL PEEK(Y,F,G,H,I,J,K,L,M):: PRG$=CHR$(F)&CHR$(G)&CHR$(H)&
CHR$(I)&CHR$(J)&CHR$(K):: LOC=256*L+M
230 IF LOC>32767 THEN LOC=LOC-65536:: LOC$=STR$(LOC)
240 IF CK$=PRG$ THEN 250 ELSE Y=Y+8:: GOTO 210
250 CALL PEEK(-31952,S1,S2):: S=256*S1+S2-65536
260 IF S<LOC THEN PRINT "PROGRAM OVERWRITTEN BY THIS ROUTINE
 , CHECK LOCATION";LOC;"BY HAND!" :: GOTO 330
270 CALL PEEK(LOC,F,G,H,I)
280 PRINT #1:CHR$(0)&CHR$(LN)&CHR$(157)&CHR$(200)&CHR$(4)&
"PEEK"&CHR$(183)&CHR$(200)&CHR$(LEN(LOC$))&LOC$&CHR$(179)&
"@1"&CHR$(179)&"@2";
290 PRINT #1:CHR$(179)&"@3"&CHR$(179)&"@4"&CHR$(182)&CHR$(0)
300 PRINT #1:CHR$(0)&CHR$(LN+1)&CHR$(132)&"@1"&CHR$(190)&
CHR$(200)&CHR$(1-(F>9)-(F>99))&STR$(F)&CHR$(187)&"@2"&CHR$(190);
310 PRINT #1:CHR$(200)&CHR$(1-(G>9)-(G>99))&STR$(G)&CHR$(187)&
"@3"&CHR$(190)&CHR$(200)&CHR$(1-(H>9)-(H>99))&STR$(H)&
CHR$(187)&"@4"&CHR$(190);
320 PRINT #1:CHR$(200)&CHR$(1-(I>9)-(I>99))&STR$(I)&
CHR$(176)&"FLAG"&CHR$(190)&CHR$(200)&CHR$(1)&"1"&CHR$(129)&
"FLAG"&CHR$(190)&CHR$(200)&CHR$(1)&"0"&CHR$(0)
330 PRINT #1:CHR$(255)&CHR$(255)
340 CLOSE #1
```

As an example, I would like to use the popular TK-WRITER program. As you go from the EDITOR to FORMATTER or back, the object file reloads. In most cases, this is not required. I say in most because, I'm not sure if the loader program will be overwritten if the buffer approaches full. Using the method mentioned, you can enter:

```
  CALL INIT :: CALL LOAD("DSK1.WRITER")
  RUN "DSK1.REFRESTORE"
```

Answer the prompt for output file with DSK1.LOADMRG. Choosing EDITOR as the check file,you find that the object file had been overwritten by the REFRESTORE program. However, the entry point of EDITOR is stated to be -1514. That's no problem, it just means a little more wo
Now do a OLD DSK1.LOAD (assuming that's what the loader is stored under. Then do a MERGE DSK1.LOADMRG. If you list the program, you wil
see parts of both routines; don't worry about that for now. Remember that entry location, lets find out what's there. In immediate mode, type:

```
   CALL INIT :: CALL LOAD("DSK1.WRITER")
   CALL PEEK(-1514,A,B,C,D):: PRINT A,B,C,D
```

The values printed will be 2, 224,248, and 142 if you have the same version I have. You now can create the check lines:

```
   6 CALL PEEK(-1514,@1,@2,@3,@4)
   7 IF @1=2 AND @2=224 AND @3=248 AND @4=142 THEN FLAG=1 ELSE FLAG= 0
```

Modify the rest of the program to do the check, then jump around the load if the check is OK and you have the new LOAD program below:
NOTE: Portions from original program by Tom Knight

```
1 CALL INIT :: CALL CLEAR
2 CALL LOAD(8194,36,244,63,232)
3 CALL LOAD(16360,85,84,73,76,73,84,250,212)
4 CALL LOAD(16368,70,79,82,77,65,84,250,132)
5 CALL LOAD(16376,69,68,73,84,79,82,250,22)
6 CALL PEEK(-1514,@1,@2,@3,@4)
7 IF @1=2 AND @2=224 AND @3=248 AND @4=142 THEN FLAG=1 ELSE FLAG=0
100 IF FLAG THEN 110
108 CALL LOAD("DSK1.WRITER")
110 DISPLAY AT(6,2):"PRESS ;": :"1 - TO LOAD EDITOR": :"   2 - TO LOA
D FORMATTER": :"   3 - TO LOAD UTILITY"
120 CALL KEY(0,K,S):: IF S=0 THEN 120 ELSE IF K<49 OR K>51 THEN 120 ELSE
K=K-48
130 ON K GOTO 140,150,160
140 CALL LINK("EDITOR")
150 CALL LINK("FORMAT")
160 CALL LINK("UTILIT")
170 END
```

Keep Your Computer in Good Shape . . .

HOW TO CONVERT ASSEMBLY PROGRAMS TO PROGRAM FORM FOR FASTER LOADING AND LESS DISK SPACE - Written by Darren Leonard, Pittsburgh Users Group, on an idea by Marty Kroll, Jr. (Reprinted from Northwest Ohio 99'er News)

If you have ever loaded an Assembly program with Editor/Assembler Option #3 you may have noticed that it takes quite a while to load. With some programs this can take over 2 minutes. These types of program are in Display/Fixed 80 format which we are going to change to PROGRAM format to load with Option #5. In addition to loading 3 to 5 times faster, programs stored in program format, i.e., Memory Image, take as little as 1/4 the disk space of D/F 80 files.

The method outlined in this article will work on 95% of all Assembly D/F 80 programs. Prior to writing this, I tried it on 20 programs, and it worked on 19 of them. It will even allow you to save an ASSEMBLY program to cassette. Thus people with an E/A and 32K can run Assembly programs!

To begin with, read page 420 of the Editor/Assembler manual. Try your program the way they outline it. If you get an error, then read on, and I will explain in detail how to get around it.

This section describes the procedure for D/F 80 files THAT DO NOT AUTOSTART! If your program does autostart, read down a few paragraphs on how to remove it with DISKO. [Ed. note - The disk sector editing program DISKO is a Fairware program in the DVUG Library.]

1) Plug in your E/A and call up TI-BASIC. Your E/A must be plugged in!

2) Type CALL INIT
CALL LOAD("DSK1.FILENAME")

3) If your program has more than one file, type in all the remaining files in order as follows:
CALL LOAD("DSK1.GAMEX1")
CALL LOAD("DSK1.GAMEX2")
CALL LOAD("DSK1.GAMEX3")
Get the idea?

4) Type CALL PEEK(8228,A,B)
PRINT A,B

5) Now 2 numbers will appear on the screen, one on the left and one in the middle of the screen. This number corresponds to the first free address in the memory which is also the last address of your program.

6) Convert this number to Hex and add A+B to come with a 4-digit hexadecimal number. Since your program normally loaded in memory from addresses >A000 - >FFD7 if you get A000 for A+B then your program has an Absolute Origin statement (AORG) and you will not be able to convert it with this method. Similarly, if A+B is A700 or smaller, then the program is loaded in an unusual manner since it canot fit in the small area from >A000 - A700. But if you come up with A+B = B000 or greater, then this method will work 99% of the time.

7) Type "BYE" and call up the Editor. Now type in the small Assembly program listed here:

```
      DEF SFIRST,SLAST,SLOAD
SFIRST EQU >A000
SLOAD  EQU >A000
SLAST  EQU >A700 (the value of A+B)
       END
```

NOTE!! PUT THE HEX NUMBER OF A+B IN THE PLACE WHERE A700 IS!!!!!

Hit (FCTN 9) twice and save to disk.

8) Load the Assembler.

For source file enter what you save in step 7.

For object file type DSK1.GAMEX4 or what you want.

Hit return for the printer output.

Type "RC" when it prompts for Assembler directives.

It will then assemble the program. You shouldn't get any errors.

9) Now load E/A Option #3.

Enter your filename DSK1.GAMEX1
DSK1.GAMEX2

Then enter the assembled filename from DSK1.GAMEX4 in 8.

10) Insert E/A disk #2 into drive one and load file "DSK1.SAVE".

Hit (ENTER) and type "SAVE" for the program n Follow the screen input prompts.

11) Now hit (FCTN +) and call up E/A Option #5 and type DSK1.YOURFILE and voila!

THE ULTIMATE SAVE

=== ======== ====

by Tom Freeman

You probably have noticed by now that loading memory image files, whether in Basic, Extended Basic, or EA #5, is MUCH faster than loading DIS/FIX 80 files. The reason is that program, or memory image, files are just that - being an 'image' of the original program in memory they can be transferred en bloc back to the RAM of the computer. Since TI uses VDP RAM for the transfer there is some limitation in the size that can be transferred in [illegible], but 48 sectors, or about 12K bytes, is still a lot more than one record in a DIS/FIX 80 file which is only about 48 bytes in a compressed file or 22 in an uncompressed one. Each record requires a DSR call and a movement of the disk drive, so you can see why these are much slower.

This is why you may want to try to convert your LOAD & RUN type files to RUN PROGRAM FILE type files, that is EA #5 instead of EA #3. What follows is a rather long article that should cover just about all possibilities for making conversions. Note that you need the <u>whole</u> file, in other words hidden files on protected disks etc cannot be converted. The first thing to do of course is make a backup copy on a fresh disk since the file will be easier to find, and you don't want to mess up your original do you?

I have to state here and now that my method makes use of DISkASSEMBLER. I have also outlined ways of converting if you don't have DISkASSEMBLER, but it is much easier if you do have it. I don't necessarily like to toot my own horn, but that's why I wrote it - to make learning and manipulating easier! I did not write it to pirate programs, as some have alleged - as a matter of fact I have yet to see a <u>complete</u> program that could be disassembled, reassembled elsewhere, and work if it was originally protected in a sophisticated manner.

Enough already!

I refer in the text to the term VDP utilities. These are the ones that are loaded by CALL INIT (or as soon as you press 3 LOAD & RUN). They consist of XMLLNK, KSCAN, VSBW, VMBW, VSBR, VMBR, VWTR, DSRLNK, LOADER, and GPLLNK. Other names that may be REF'd are addresses that are resolved by the loader, or DEF's in other programs and also resolved, so you don't have to worry about them.

Here goes...

<u>1) You have the source code</u>

This one is easy! Just make sure that there is no AORG in the >2000 to >4000 range. Now, unless they are already there, insert the following:

```
       DEF  SFIRST,SLAST,SLOAD
SFIRST
SLOAD  B    @START
```

START, or whatever you have labelled it, is where the program actually begins. Also, at the end of the file, where you see the END directive, put the label SLAST at the beginning of the line. Also make sure that the auto-start feature is not activated by the presence of the START label after the END directive.

Now reassemble using the R option if necessary, and C for speed of loading. Next proceed on to step 5) below.

<u>2) No source code</u>

<u>A) With DISkASSEMBLER</u>

Run DISkASSEMBLER on the DIS/FIX 80 file you wish to convert. You will get all the information you need: whether the file is absolute or relocatable, compressed or uncompressed, the range of addresses used, and the names of all REFs and DEFs, as well as whether there was mixed AORG and RORG code, or out of order code. In the latter two cases there may be some difficulty in changing to memory image format. See NOTE 2 below.

Note down the first and last addresses. If the file is RORG, add >A000 to each. Note whether there are REFs to the VDP utilities, in which case see NOTE 1 below. And lastly note whether there is an auto start or not. In this case, if the file is compressed, go on to D) below. If it is uncompressed then load it into the TIW or EA editor. Scan down to the end where a line begins with a 1xxxx or 2xxxx and delete this line. Then resave the file (in TIW, use PF, then F DSKx.filename - in EA, 3 SAVE, N for DIS/VAR 80? prompt). Now go to 3)

B) Without DISkASSEMBLER

Load the DIS/FIX 8Ø file into the TIW or EA editor. You may get an error message "control characters removed" in EA but don't worry about that just yet. Just press enter then 2 EDIT. If you see lots of blank spaces in EA or control characters in TIW then the file is compressed. Some of the work you do will have to be done with a sector editor such as Advanced Diagnostics or DISK+AID but while you're here scan down to the end of the file where you see a : at the beginning of a line. This is the end of the file, and is preceded by any external REFs and DEFs with readable blocks of 6 characters (spacing always pads the name to 6). If you see any names of VDP utilities you will have to prepare a special file covered in NOTE 1 below. Note whether SFIRST, SLAST, and SLOAD have been defined here. Examine the line above these or the one above the : if there were none. If it begins with a 1 or a 2 then this is an auto start file and will have to be modified. If the file is uncompressed delete the line then resave it (in TIW, use PF, then F DSK1.filename, in EA, 3 SAVE, N for DIS/VAR 8Ø? prompt). If the file is compressed go on to C).

Now go back to the file if it is uncompressed and return to the first line. You will see a Ø followed by 4 characters which are the ASCII representation of the number of bytes of relocatable code in Hex. This is followed by an identifier of 8 characters (it may be spaces, or padded to 8 with spaces). In columns 14-18 you will see either 9xxxx or AØØØØ. In the first case the code is absolute origin at address xxxx. In the second it is relocatable and will load at AØØØ. Note down the value in either case. If the code is relocatable you may add the hex number that followed the Ø at the start of the line to >AØØØ to obtain the last address used. For absolute code scan down the beginning of each line. Each should begin with a 9yyyy where yyyy is the start address of that line. When you get to the last line of code you have almost the last address. Just add to it 2 for each group of 5 characters after the 9yyyy until you get to a 7F near the end of the line. You now have the first and last addresses which will be used below.

C) Compressed file -Address Range- No DISkASSEMBLER

On your backup disk find the first sector (it should be 34, or 32 in older MYARC FDC's). Remember that each "line" or record begins on byte Ø, 8Ø, or 16Ø of a sector. The first line should start (in Hex not ASCII) 3[illegible] and then 16 characters which are the identifier (they are readable in ASCII, as 8 characters). xxxx represents the number of bytes of relocatable code. Note it down. After the identifier you will see 39yyyy if the code is AORG or 41yyyy if it is RORG. Write yyyy down if the code is AORG, or AØØØ if it is RORG. If the file is RORG you may add the xxxx found at the start of the line to AØØØ to get the last address. Otherwise scan down sector by sector until you get to the last line before the REFs and DEFs or auto start, in other words the last line beginning with a 39 in Hex. The next line will start either with a hex number from 31 to 36 or with a 3A (Hex for :). This is the first address of the last line of code. Now add 2 to the yyyy after the 39 for each group of 6 characters until you get to the 46 at the end of the "line." You now have the last AORG address. Write down the first and last addresses whether AORG or RORG - they will be used below. Note there are a few strange files that apparently were assembled with a different assembler from the one TI supplied us, and each line does not begin with an address. In these it will be almost impossible to determine the last address without DISkASSEMBLER.

D) Compressed file -auto start- No DISkASSEMBLER

With your sector editor go to the last sector or two and find the line that begins (in Hex) 31xxxx or 32xxxx. Change the 31 or 32 to 46 and write it to the disk.

E) Other auto starts

A few sneaky programmers auto start their programs not with the above method but instead by inserting the start address into the user interrupt hook at >83C4. If you have a file that auto starts but can't find the 1 or 2 (in ASCII, 31 or 32 in Hex) this may be the method. Look at the end of the last line of code for the following: (compressed) 39 83 C4 42 xx xx, (uncompressed) 983C4Bxxxx. If you see it xxxx is the start address. Replace the 39 with a 46 if the file is compressed, or the 983C4 with 8FØØØ if it is uncompressed, and th program won't auto start anymore. It may not start at all, but that doesn't matter because we don't want it to! We just want to load it, then convert it.

3) The First Executable Instruction

For the EA #5 loader to work (and all loaders based on it) the initial code must be an executable instruction. If you have a file already in memory image format you can examine the code after the first 6 bytes and see what I mean. It frequently is a B @>xxxx (Ø46Ø xxxx is the actual code) or JMP xxxx (1Øxx) where xxxx is the actual start of the program. Or it may be a normal sequence of code e.g. MOV 11,@>xxxx LWPI yyyy etc. indicating the programmer anticipated saving in this format. If your file does not begin this way there will have to be some additional preparation.

If your file is auto start and you have determined it is done by one of the two methods above then you know what the start address is. If the file is not auto start you should know from the instructions for the program what the name of the start address is (for RUN after LOAD, or CALL LINK in EA Basic) and you can look for it at the end of the file (with a sector editor if the file is compressed). You would see something like 5xxxxSTART (uncompressed) or 35xxxx5354415154 (compressed, reading in Hex). These are relocatable start addresses and the xxxx should be added to AØØØ. Absolute address have a 6 or 26 before the xxxx. If you used DISkASSEMBLER the start address was displayed for you. NOW see whether the start address is also the first address of the file. If it is you are in luck and may proceed on to the next step. If not you may still proceed, but when you are finished see NOTE 3 because further modifications are to be made.

4) SFIRST SLAST file

If your file already has SFIRST, SLAST, and SLOAD in DEFs in it, the programmer anticipated using this method and you may go on to step 5). If only one or two of these names are used check to make sure that they are the first, last, and first addresses respectively. If they are, then eliminate the appropriate ones from the file below. If not use a sector editor to change any letter in the name (and type 8 over the 7 at the end if the file is uncompressed).

Now prepare the following special file, using the EA editor. Lines 2-4 should abut the left margin.

```
       DEF SFIRST,SLAST,SLOAD
SFIRST EQU  >xxxx
SLOAD  EQU  >xxxx
SLAST  EQU  >yyyy
       END
```

Here xxxx and yyyy are the first and last addresses determined above. Save the file in DIS/VAR 8Ø format, then go to the assembler and assemble it using the file just saved as source file and a different name for object code. For List File and Options just press enter. You should rapidly get the ØØØØ errors message.

5) The Reassembly

Now that you are all prepared the final job is easy. Using the 3 LOAD & RUN option of EA load your file (modified if necessary to eliminate the auto start), the file prepared in 4) if it was needed, and SAVE from the [illegible] disk. When the cursor appears again, press enter, type in SAVE for program name, then follow the screen prompts. For purposes of neatness choose a name for the output file that ends in a 1, since 33 sector blocks will be created and each successive one will have the last character increased by 1.

The newly created file should run in EA #5. It won't if there were REFs to the VDP utilities, or if the actual addresses were inserted in the original source code. In this case, see NOTE 1 at the end of the article.

Please note that the file you are converting should either be all RORG in which case it will load at >AØØØ, or it should AORG at >AØØØ or higher. If it AORGs in the >2ØØØ to >4ØØØ range and above >AØØØ you should save the two parts separately (create two files in 4) above) using a file name ending in 2 for part above >AØØØ. Furthermore the range >2ØØØ to >2FBØ cannot be used since this is where SAVE loads, unless that area is really only a BSS type block. In DISkASSEMBLER this would be indicated by a series of AORGs without DATA. If you are examining the actual DIS/FIX 8Ø file in the editor or with a sector editor you would have to see 39xxxx [illegible] in Hex) or 9xxxx (uncompressed, in ASCII) [illegible] you past >2FBØ with no 42's or B's in between for this to be true. In this case SAVE itself would be saved, and overwritten when the program runs, but that is OK because it isn't needed anymore. Furthermore the program can't use the range between >2ØØØ and >2676 because the EA loader and utilities reside here. If the program appears to do that it was meant to be loaded by some other loader, such as Mini Memory, so something else will have to be done.

All these problems can be fixed up if you have the SAVE source code since it can be AORG'd wherever you want it, and therefore not interfere with the original program. If you have DISkASSEMBLER this can be done by following the instructions in the appendix (naturally I would love it if you would buy a copy!) For those that don't have it I am placing the source code in the club library. The disk can be purchased for $5.ØØ - all proceeds to the Club, not me! You then place an AORG in it that gives you >8ØØ bytes outside the range of the program to be converted and reassemble it. If the file to be converted is to be in the >2ØØØ range then you must use the Mini Memory cartridge to load it, and an RORG assembled SAVE can be used.

NOTE 1 The EA Utilities

Normally EA #5 programs should stand alone since the utilities are not loaded in first, as they are with #3 type files. There is a way around this however. If the file had REFs to the VDP utilities you know this will be necessary. If there weren't such REFs but the converted program won't run then perhaps there were uses of the actual addresses in the program and you can try this method.

Prepare a short file as in step 4) using >2ØØØ as xxxx and >2676 as yyyy. Assemble it then proceed to step 5) and use a filename such as UTILØ as the output file. Find the file on disk with your sector editor, and change the first two bytes from ØØØØ to FFFF. This is a generic file and may be used with all converted programs that need it - all you have to do is copy it to the disk containing the converted program and change the name to one the same as the others but with the last character decreased by 1. I prepared DISK+AID in this manner; the converted files are called DISKAIDØ, DISKAID1, and DISKAID2.

If your file contained code between >2676 and >4ØØØ that either didn't interfere with SAVE, or you used a modified SAVE, then you could save it together with the VDP utilities. However this is not necessary - you would just have two shorter files, and waste 1 or 2 sectors.

NOTE 2 High and Low Memory mix

If the file to be converted contains code below >4ØØØ and above >AØØØ you need to convert the two parts separately, using the relocatable SAVE if necessary. If there is a low mem piece AND the utilities are also needed then for convenience you may want to save the entire low mem block together even though some space may be wasted. You may also do it in 2 separate pieces if you wish. In any case change the first 2 bytes of the file(s) to FFFF. Also remember that if you have 2 files, the second must have the last character increased by 1 (and again if there is code above >AØØØ).

NOTE 3 Special File for Executable Instruction

If your first file created DID have an executable instruction at the beginning, AND you needed the special [illegible] utilities file, then change the latter to have a name AFTER the program files, and change its first two bytes back to >0000. Then you are set. If there wasn't an executable instruction and you do need the VDP file, then change bytes 7-10 of the VDP file to 0460xxxx where xxxx is where your program actually starts. You can do this because those 4 bytes were actually what CALL INIT loaded at >2000 to >2003 and aren't needed. One last case where you don't have the first executable instruction but don't have to make an extra file is where there was a BSS of at least 4 bytes at the start of the file (in other words, successive origins in the DIS/FIX 80 file). Then you can replace bytes 7-10 of your first program file with 0460 xxxx.

If none of these special cases obtain, then you will have to prepare this special file. It actually is rather easy. Find an unused area of memory, either in low mem between >3F50 and >4000, or high in the high mem, above >E000. Write the following source code:

```
        DEF   SFIRST,SLAST,SLOAD
        AORG  >F000   OR WHEREVER YOU HAVE DECIDED IT GOES
SLOAD
SFIRST  B     @>XXXX  XXXX IS YOUR ACTUAL START ADDRESS
SLAST   END
```

Save this file, then assemble it, load it, then load SAVE, then press enter, type in SAVE, enter and follow the screen prompts. Use as your file name one with the last character one less than your previous first file. You will create a tiny 2 sector file which SA will find the start address in. Remember to use a sector editor to change the first 2 bytes from 0000 to FFFF.

NOTE 4 Multiple Files

If your program actually contained multiple files to load before the CALL LINK or the entry of program name, the instruction above still apply, but it may be a little harder to find the information you need. I'll be happy to help if I can, but you should be able to do it. Remember to load ALL of the files before running SAVE.

This article wound up a LOT longer than I intended. Unfortunately I have never been accused of being to brief. However I was really trying to cover all possibililites. I hope it works for you every time! Enjoy.

Editor's Note: Tom Freeman is a practicing pediatrician in the Los Angeles area and a regular (prolific) contributor to the excellent "LA Times", newsletter of the Los Angeles Area 99er Users Group. He is also author of the terrific "DISkASSEMBLER" software available from MG (1475 W. Cypress Avenue, San Dimas California; $19.95 + $2 S & H). He is skilled in both assembly language language and GPL ("Graphics Programming Language") and continues to produce innovative public domain routines as well as his commercial efforts - the most notable being his two-column and "quad-column" print routines.

Some Help When You Need It
"C" Tutorials

The C Language and You

By Warren Agee
Compuserve ID 70277,2063

The TI-99/4A is getting to be quite an O-L-D computer! But despite its age, quite a bit of software that is commonplace for other machines has yet to surface for the 99 enthusiast. One of those goodies is a C compiler - the language which is currently the rage the of newest and brightest computers in the market today. But the wait is over! In or around September 1985 a gifted systems programmer from Ontario, Canada, Clint Pulley, filled a deep whole...a C compiler for the 994A!

But what is C? C is a language that was developed by Dennis Ritchie on the Unix operating system on the DEC PDP-11. Since then various versions of the language have popped up on almost all personal computers. In fact, it is the language of choice for the newest breed of personal computers -- the 68000 machines likethe Amiga and Atari 520ST. C's long list of strengths includes the fact that it is not tied to any one operating system and machine, which makes C code rather portable. This doesn't mean that a program written on a Macintosh will run on an Atari 520ST, but it does mean that the process of converting such a program over to a new machine is greatly simplified. C is also relatively small, it can be learned quickly. It is a relatively "low level" language, which means the programmer has more direct control over his work and the machine. However, this facet also has its drawbacks: the programmer has to be more careful in what he does and has to have a good understanding of how the machine works.

C is not a language for beginners, mainly because it is a low-level language. But it *is* much easier to learn and use than Assembly, and perhaps easier for some than FORTH. The most significant advantage to this language is that it allows people without the knowledge or expertise (or sanity??) to program in Assembly can now produce high-quality, fast software that in many cases rivals assembly.

Perhaps I should correct myself and say we now have a "c" compiler, not a "C" compiler. What? That's right, a little "c". You see, the compiler that Clint wrote, called c99, really supports only a subset of the full C language, often referred to as K&R, which stands for Kernighan and Ritchie, the creators of the language. This is due to the memory constraints of the 99/4A. The C language was developed on a mainframe, not on a 48K home computer. This means that many compromises had to be made in order to squeeze a functional C compiler into the 4A. Nonetheless, c99 is a very capable language that stands by itself just fine.

The C language is different from BASIC in that it is compiled, which means you key-in your programs with a word processor, then run them through the compiler. This program reads in your source code and generates assembly language code, which is the finished program, which can be loaded in separately and run. The mechanics of creating a program with c99 differ from most compilers on other machines in that the c99 compiler does not generate the finished program. It's really a two-step process. The compiler generates assembler source code instead of object code. The resultant file is then run through the 99/4A assembler,which comes with the Editor Assembler cartridge. So as you can well imagine, you need the E/A cartridge in order to program in c99! However, a thorough knowledge of assembly language is in no way a prerequisite to programming in C. But one does have to know how to work the assembler, which is not hard at all.

But what is all the fuss over the C language? Who cares if it's compiled? I care. A lot of people care. So stop asking questions and listen. The singlemost important advantage of a compiled language (like C) is >>>>>SPEED<<<<<. Zooooom...the only thing faster than a c99 program is an assembly language program. Not even FORTH can beat it. C is also much easier to learn than assembly. It is easier to read than assembly. Its easier to go back and modify after time than assembly. So let's all pitch assembly out the window! No, we must not do that, because there is one major drawback of c99...it tends to create "bulky" programs. If one were to write a program that prints mailing labels in both languages and compare, you would find that it probably took less time to write it in c99. It probably also compares favorably to assembly in its speed. But the size of the programs will be dramatically different...assembly is much much more compact. This is very important to people like us who only have 48K of memory with which to work!

However, in all honesty, its not that bad. I have been able to write functional, effective programs in c99 that just fit into 48K. You may not be able to port Lotus 123 or dBASE III, but you can sling some mean code if you stay on your toes. Fortunately, some very nice people have made that job easier on us, namely Clint Pulley, Tom Wible, and Richard Roseen, who have developed "optimizers." These doo-dads compress your program, allowing them to fit in a smaller space, therefore making more memory available to you. Clint wrote the original c99 optimizer, and Tom and Richard continue to enhance it.

Speaking of enhancements, Clint Pulley seems very dedicated to his project. He is constantly updating and upgrading his compiler to bring it up to snuff with "the mainstream." Although at the start c99 was more a novelty than anything else, Clint has raised the power and versatility of c99 to a level of commercial quality. As of this writing, I know of three commercial programs soon to be available that are written in c99, and I have no doubt that more is on the way.

[Editor's Note: Warren is well-qualified to write about c99. He is one of the very first to write a commercial program using the language, "Total Filer" from Asgard Software, P.O. Box 10306, Rockville, MD 20850.]

c99 Beginner's Tutorial #1

by Ron Albright
Compuserve ID 75166,2473

I have been exploring c99 for the TI of late. Written by Clint Pulley (38 Townsend Avenue, Burlington, Ontario, Canada L7T 1Y6) and available as Fairware, the language is a full-featured version of "small c". I have found few limitations with the language (lack of floating-point and math routines are the major ones), and have been able to do some nice routines with the language. Briefly, C is a very popular programming language through which, it has been estimated, 70% of commercial software for other machines is written. So what makes it different? It is a "compiled" language. That means, once you have written your program in c99, you run a companion program called a compiler. The compiler takes your C source code and generates assembly source code. The resultant code can then be run through the TI Assembler to generate object code, which executes just as fast as if you went through the strenuous (to me, anyway) task of writing assembly source code to start with. C is much easier to learn that Assembly language and is efficiently compiled with the c99 compiler. I have seen some programs written with c99 alone (there are a few on Compuserve; a simple text editor and a word-counter for TI Writer files by Warren Agee, a program similar to the TI Writer formatter, and a graphics demo by yours truly) and they are indistinguishable from pure assembly language, because the end-product is just that. If there is any interest, I will address the language more in depth in some more starter-level tutorials. I am no expert,by any stretch of the imagination, but I am learning and plan to spend a great deal of time with the language. It is a marvelous programming tool and, hopefully, this simple file will help you get started. Learning a new language is never easy, but it is time we all advanced beyond BASIC and started working in another environment. c99 provides a reasonable alternative. I could never think in reverse, so I gave up on Forth; I am too dense to learn assembly language. Pilot is too slow and requires too many disk accesses. Besides C is used in so many other machines and for so many other applications, it has to be good. Let's begin by seeing what we have to work with.

First, equipment-wise, you need the following: console, monitor, 32K memory expansion, at least one disk drive and controller, the Editor/Assembler package (cartridge or disk version) and, of course, the c99 system disk. A printer is nice (see below) but is certainly not imperative for programming purposes. Ideally, you would have two drives as this makes the work much easier, as does having at least double-sided drives (but ain't that always the case!). If you have double-sided drives, you can save yourself a lot of disk-swapping by, first, of course, making a backup of the c99 system disk and, secondly, copying from the Editor/Assembler disk, the files ASSM1, ASSM2 (the files for assembling source code) and EDIT1 (for the E/A Editor) on to the c99 system disk. But, if you have a single-drive or single-sided system, don't despair...things will work just fine with what you have.

Once you have gathered your tools, you should get a disk directory printout of the c99 system disk. Pulley even provides a disk catalog program on the system disk (called "SD" and running out of E/A 5 on my disk) but it doesn't print to printer). You will notice that there are a long of files in all shapes and "colors" (D/V 80, D/F 80, and PROGRAM files) and we will first go over what is important and what is not. Some of the files you will be using a lot, others seldom if at all, at least to start. Here are some of the files you should have and what they are for. I will list them in order of importance and probably frequency of use.

C99C,C99D,C99E

These are the compiler files. They are the heart and soul of the c99 system. There are PROGRAM image files and are run from Editor/Assembler option 5. Unlike some PROGRAM image files, these CANNOT be run from option 3 of the TI Writer module. In my brief experiment they could not be loaded from XB with the FUN LOADER from Australia. The first thing I did with these files is rename them to be UTIL1, UTIL2, UTIL3. Then, when you chose the LOAD and RUN option from E/A (option 5), you only have to hit "Enter" and the files will be loaded by that name as a default without typing them in.

CSUP

This file is very important. It is a D/F 80 (which always means it runs from E/A option 3) which must be loaded immediately after you load you completed, assembled program. We will discuss this more later, but suffice it to say that your c99 program will never run if you don't load this file after it and with it.

C99MAN1,C99MAN2,C99MAN3

These are the D/V 80 files that contain the documentation Clint Pulley provides with the c99 system. They are not going to go very far in teaching you how to program in c99. Like the manual TI provided with the TI Forth system, they are simple a brief tutorial on how the different files work, and what they do, what the error messages mean, ect. They are quite adequate for their intended purposes. Pulley tells you up front "This manual assumes a knowledge of standard C or the availability of a suitable reference." That translates into "If you have never programmed in C, go buy a book!" I will recommend a couple at the end of this piece. Far enough, Clint! If you have a printer, print these files out for future reference. If not, find a friend who does. You will need a hard-copy of these files.

C99ERRORS

This is a short D/V 80 file that contains a listing of the 30 or so error messages that the compiler will embed in your compiled code when it encounters one. It will only embed the error number. You will have to look in this file to find out what the number means. Print this out also.

C99SPECS

A terribly important D/V 80 file. This short file tells you what c99 supports and, more importantly, what it does not support, when compared to standard C. Why is this important? I have yet to find a book that addresses only "small c", the version of C (more limited than "big C") that c99 is modeled after. All the texts I am aware of cover the full C language. Small c and c99 do not have all the functions of C. When you look at program listings out of these texts, you will quickly become frustrated if you try to type them in verbatim as they are already. Many program statements in C will give you errors in c99. You have to study this file when typing in program listings out of books to avoid these errors. For example, C supports "floating-point" arithmetic; small c and c99 do not. There are other examples covered in this file; print it out. You will need it.

GRF1DOCS

This is the documentation for the graphics routines supported by the current version (1.32) of c99. Print it out.

ERRFIND1

This is a helpful file provided by Clint. It is a PROGRAM file to be run out of E/A 5. Run this if you have run the c99 compiler on a source code file and received the dreaded "!!ERRORS!!" message. What it will do is prompt you for the compiled file's name (not the original source file!), read it in very quickly showing the file on the screen as it reads it. You can stop to read the file by holding down any key; releasing the key resumes the read. Then, after it has read the file, it will flash the lines again on the screen that contain the error message so you can (1) see where the error occurred and (2) what the error message was. It is also nifty for reading ANY D/V 80 text file. It's purpose, though, was to help in debugging.

There are several other files that are, for the most part, files to be included in your c99 source codes as you use certain functions. We will go into this in some depth later, but you will use an "#include dsk1.filename" in your source files to copy these files into your source codes. For example, if you used some graphics commands in your source file to draw some sprites or such, you would need to use "#include dsk1.grf1refs" in your source code as a line before you started using the graphics commands. Else, the compiler won't understand what they mean and give you a multitude of errors. If you use commands to access disk files, you would have to use "#include dsk1.stdio" (for "standard input and output") before you started opening and reading from disk files. Notice the use of lower case in these #include statements. The compiler can use lower case, unlike the E/A Assembler which only accepts upper-case. Just keep the list of the other files as they will be used as you start to type in programs.

How does one enter programs with c99? You can do it two ways. You can use TI Writer, but always use "PF" to disk rather than "SF" and throw in the "C DSKx.filename" syntax to clean all the control characters out. Or, preferably, you can use the Editor of Editor/Assembler. We won't do a program this time, as you have enough to do for now.

What about recommended books? I strongly recommend "C PRIMER PLUS" by Waite, Prata and Martin (Sam's Publishing, 1984). It is 500 pages and costs about $22. It is the "Going Forth" (Brodie) for C. It is easy to read, starts at a beginner's level and is chock full of example programs. Some usable with out dialect of small c, some not (at least without some conversions). I went though two other books on C before I found this tome. It is the best I have seen. If you know C, the bible (but much too advanced for me) is "THE C PROGRAMMING LANGUAGE" by Kernighan and Ritchie (Prentice-Hall, 1978). I found a back issue of Byte magazine also useful. The August, 1983 issue is devoted to C and contains some very nice articles and tutorials. You can still get a copy of this from Byte.

c99 Beginner's Tutorial #2

by Ron Albright
Compuserve ID 75166,2473

Last time we touched on what c99 is, and what files come on the disk and what some of the more important ones do. This time, we'll actually do some code. As we progress, we will stress some sort of style in how we enter programs. I am no expert on style (or c99, for that matter), but since c99 is so free-form and has no line numbers to follow, it can be very difficult to read programs if you don't follow some rules. These rules are not universally agreed upon, but we'll try to develop some sort of easy to read style of our own. I will make a few assumptions to start. First, I will assume you have a single-drive system with only single-sided capability. Second, I will assume that you have a basic understanding of the Editor/Assembler package, i.e. you know how to use the Editor, and run programs out of either option 3 or option 5. I will, further, assume, that you have assembled at least one source code file with E/A. If these assumptions are incorrect, let me know and we'll touch on the Editor/Assembler more next time. Let's get started.

Take a clean disk and copy the following c99 files onto it:

CSUP	D/F 80	12 Sectors
PRINTF	D/F 80	14 Sectors
UTIL1	PROGRAM	33 Sectors
UTIL2	PROGRAM	33 Sectors
UTIL3	PROGRAM	29 Sectors

Next, from the Editor/Assembler disk, copy these files to the same disk:

ASSM1	PROGRAM	33 Sectors
ASSM2	PROGRAM	20 Sectors
EDIT1	PROGRAM	25 Sectors

If my addition is correct, that gives us 199 sectors on our work disk. Now we are ready to proceed. Keep our work disk in the drive and insert the Editor/Assembler cartridge. From the menu, load the Editor and go into the Edit mode.

Type in this program.

```
/* c99 The smallest c99 program */

main() /* a comment */

{
   /* we aren't going to do anything! */

}
```

Congratulations! You have just entered your first, valid c99 program. Let's look at it. The first line is nothing more than a "REM" statement. Instead of REM, c99 recognizes anything enclosed within "/*   */" as a comment and ignores it when compiling. You can put anything between these comment delimiters, and it will survive compiling without error. Use them frequently as you program. As we mentioned, c99 programs are difficult to read at best and REM statements are useful to remind yourself, as well as other reading the program, what you had in mind. As shown on the next program line, the can also be used on the same line as compilable code, so comment each step of your code for clarity. A routine called "main" is required somewhere in each and every c99 program. Typically, it is the first block of code, sets things up, and calls the other routine(s) to take over. When the compiler sees "main()" (or anything with the "()" after it - like "first()", "setup()" - it labels this as a function; a subroutine in Extended Basic. A string of functions make up a program. They are just like you were using "SUB routine" in XB. It is run when its name ("main", "first", "setup") is "called". The "main" routine is run whether it is called or not (guess that is why they call it "main").) is called. More on this later. But, for now, thing of c99 as simply a series of "calls" to blocks of modular code called functions and a function is labeled with "name()".

Each function is enclosed with a pair of braces - it starts with an open brace ({) and ends with a closed brace (}). This tells the compiler where this block of code starts and ends. Everything within those braces is part of that function. In our first program, the only thing in the main function is a "REM" statement, so it will "do" nothing. But it is compilable. A function may include a call for another function. Look at this:

```
main()
{
     doit();
}

/* doit doesn't do anything! */

doit()
{
/* see! Nothing here to do! */
}
```

This time, main calls up the second function, "doit" which, also, doesn't do anything. But you can see how programs are built. Typically (but not necessarily) the main function will include all the calls to the functions that make up a whole c99 program. Its like having an XB program that is nothing more than a series of "GOSUB"s (really, a series of "CALL SUB routines). Each function call doing its task and returning control back to the main, or controlling program. The good c99 program will break large programs into smaller ones and write a function for each. If a function can stand alone (has nothing in it unique to a single program) the programmer eventually develops a "toolbox" of useful small routines (functions) that can be combined in different ways to solve problems. That is just one of the beauties of c99.

So, let's compile this program. After typing it in, hit FCTN 9 twice, get the EDITOR menu and elect to save it to disk. Your main work disk should have plenty of room, so no disk swapping. After saving to disk 1, hit FCTN 9 again, and get the main E/A menu. Chose Option 5 to "RUN PROGRAM FILE". The three compiler files, which I have renamed UTIL1, UTIL2, UTIL3, run out of option 5, not option 3 (which runs D/F80 files). When you are prompted for "Program Name:", since you have changed the name of your compiler files to UTIL1-3, you only have to hit enter. The default name for E/A 5 is UTIL1 and those files will then be loaded automatically (now you see why I renamed them). You will then be prompted by the c99 compiler (prompts will vary depending on which version of c99 you use) for a input file name. Type "DSK1.filename" (filename being generic for whatever you called the file you typed in and saved to disk). You will then be prompted for an output file name. Call it "filename/C", just to remind yourself that is a compiled file. Then, hit enter and you are off and running. The compiler will flash each function name on the screen as it is compiled to show you where you are in the program. You should see only "main" if you are compiling the first program, and "main", then "doit" if you are compiling the second routine. If an error is encountered, you will be told. But we'll assume you typed these short routines in without error for now. It shouldn't take long and you are told to press enter to continue after the compiler is finished.

Now what? If you catalog your disk now, you should see the initial source code file you typed in and saved, and now a second file called "filename/C". Both should be D/V80. You have one more step to do before you can run the program. What the compiler produced was assembly language source code. Like all source code, it has to be assembled. Get to the main E/A menu and choose Option 2, Assemble. When asked to "Load Assembler?", hit "Y", and since we put the E/A assembler files on disk 1 (ASSM1 and ASSM2) they should load right in without swapping disks. You are then prompted for the "Source File Name". Type in "DSK1.filename/C" (NOT the program you typed in and saved, but the compiler's output filename). For an "Output File Name", I use "DSK1.filename/O" to let me know this is object code. Then hit enter for each of the next two assembler prompts ("List File Name" and "Options"). The assembler should start right up and finish with the assembly process. Now, catalog you disk again. You should see a third file added now - "filename/O". This time, it is not D/V80, but D/F80. Assembly language OBJECT code. You have produced an assembly language program. How do you run this "do nothing" program you have written? Go back to the main E/A menu again. Choose Option 3 from the menu. When asked for "File Name", type in "DSK1.filename/O". Then hit enter. You get the same prompt again ("File Name:"). This time, type "DSK1.CSUP". This "c99 Support" file MUST be loaded after you load ANY c99 program. Hit enter. When you get the prompt for the third filename, just hit enter this time. When asked for the "Program Name", type in "START". All c99 programs run with the program name start. Your do nothing, super-duper assembly language program should now "run". You then immediately get the "hit enter to continue" message and you have finished.

Well, how does it feel to have generated a assembly language program just like the "big boys"? Next time, we will do something with a little more substance. We will create a simple menu, which will demonstrate keyboard input and the "printf", "puts", and "getchar" functions. But, for now, I just wanted to go through the mechanics of running the c99 system. Till the next tutorial, get a C book, read the "manual that comes with c99 itself, send for the new version 2.0 of the compiler, and if you haven't paid Clint do so.

c99 Beginner's Tutorial #3

by Ron Albright
Compuserve ID 75166,2473

When I started to learn BASIC (and later, Extended Basic), I remember how I did it. I first typed in other programs from magazines and books. Then I started to do my own programs. And the first type of commands I used were the graphics commands. I sure didn't jump in with file handling or string manipulation! Anyway, I found myself doing the same thing with c99. I typed in some programs out of a book, then started playing with my own routines with graphics. Then I tackled a game. I have though all long that is you can learn the logic involved in a game, you have learned a great deal about the programming structure of a particular language.

In this tutorial, we will try to accomplish a couple of things. First, a glimpse at some of the graphics commands available to c99 in the "grflrf" library (that comes with all version 1.32 or higher), and, secondly, a look at how to convert a short BASIC graphics display to c99. It really isn't that hard.

Listing 1, below, is a short BASIC program from Ed York that has appeared in several UG newsletters. It is a colorful graphics display. Listing 2 is a conversion of the program to c99, done by me. They both accomplish the same thing graphically. I have commented the c99 source code to try and explain step-by-step what we did. I think as you look at the programs, you will see how similar both the graphics commands and the logic is between c99 and BASIC. It is, to me, much closer to BASIC than Forth was. See if you agree.

Listing 1

```
100 REM  COLOR BONANZA
110 REM  WRITTEN BY:
120 REM  ED YORK
130 CALL CLEAR
140 FOR A=40 TO 136 STEP 8
150 CALL CHAR(A,"55AA55AA55AA55AA")
160 NEXT A
170 FOR B=2 TO 14
180 CALL COLOR(B,1,1)
190 CALL VCHAR(1,2*B,24+8*B,22)
200 CALL VCHAR(1,2*B+1,24+8*B,22)
210 NEXT B
220 FOR C=2 TO 14
230 CALL SCREEN(INT(16*RND)+1)
240 FOR D=2 TO 14
250 CALL COLOR(D,D,C)
260 NEXT D
270 CALL KEY(0,E,F)
280 IF F<1 THEN 270
290 NEXT C
300 GOTO 220
```

Listing 2

```
/* COLOR BONANZA   This and the next 2 lines are REM's  (line 100) */
/* WRITTEN BY:     (110) */
/* ED YORK        (120) */

#include dsk1.grf1rf     /* required to use the graphics commands */
#include dsk1.random;c   /* required to use the random number commands */

main()
{
       int a,b;     /* MUST declare ALL variables used in a routine at start */
       grf1();      /* MUST be used as first command for graphics library use */
       clear();     /* Same as CALL CLEAR (130) */
       randomize();/* Same as RANDOMIZE in BASIC */
       for(a=40;a<=136;a=a+8)  /* Lines 140 and 160 ALL IN ONE STATEMENT! */
            chrdef(a,"55aa55aa55aa55aa"); /* CALL CHAR in line 150 */

       for(b=2;b<=14;b++) /* Another FOR-NEXT loop -lines 170 and 210 in one */
            {             /* Multiple lines in for loops need to be braced */
            color(b,1,1); /* Same as CALL COLOR -  line 180 */
            vchar(1,2*b,24+8*b,22); /* Just a plain old CALL VCHAR! line 190 */
            vchar(1,2*b+1,24+8*b,22); /* line 200 */
            }             /* Closed braces after FOR LOOP */
       fun();             /* Gets a little tricky here. Since there was a
                             "GOTO" statement in line 300, I decided to make
                             a new routine starting at where the GOTO directs
                             the BASIC program - line 220. That way, I can call
                             the second function from itself, in essence,
                             creating a "GOTO". See below. Anyway, that is why
                             I started a new function called "FUN()". I call it
                             from the Main() routine here by just calling the
                             name of the routine. Its just like I said GOSUB
                             or, in XB, had created a user-defined SUB FUN and,
                             here, said CALL SUB FUN. */
}

fun()      /* Start of a new function */
{          /* All functions start with an open brace */
       int c,d;  /* Declare these variables at the start!!! */
       for(c=2;c<=14;c++) /* start of another FOR loop-lines 220,290 in one! */
            {             /* multiple lines after a FOR need to be braced! */
            screen(rnd(16)+1); /* CALL SCREEN in line 230 */
                 for(d=1;d<=14;d++) /* Start of a nested FOR LOOP - line 240 */
                    color(d,d,c);   /* CALL COLOR in line 250 */
            getchar(); /* Just waits for a key to be pressed - lines 270,280 */
            }          /* Close that brace for the FOR loop */
       fun();          /* See that GOTO 220 in the BASIC program? This is the
                          same thing - it just keeps calling "fun()" which is
                          nothing more than the program starting at line 220.
                          so, by separating the lines where the GOTO starts
                          into a separate routine, we can now call it over and
                          every time we would be using the GOTO in Basic. */


}                      /* Close braces for fun() routine */
```

Notes:

[1] Compile the program with the Compiler. You must have version 2.0 of the Compiler to use the "FOR" statements. Make sure the D/V 80 file "RANDOM;C" and "GRF1RF" is on disk 1. The assemble the output file. Then, load the assembler output (which should be a D/F 80 file), then from E/A option 3 still load the file "CSUP" (another D/F 80 file) and "GRF1" a third D/F 80 file. Then hit enter and use the program name "START". It should run.

[2] The only complicated move was separating lines 220 through line 300 into the separate function "fun()". This was done because line 300 in the BASIC program is a GOTO 220. Since there is not GOTO function in c99, we separate out those lines and use recursion in "fun()". Recursion simply means a routine calls itself over and over, just like a GOTO. I hope you can follow this.

[3] We could have used a function similar to CALL KEY(0,E,F) as in line 270. But, by using "getchar()" we accomplish the same thing in one line. Getchar waits for a keypress automatically without testing for "status".

[4] FOR-NEXT loops in c99 are three parts. Just as

```
240 FOR D=2 TO 14
250 CALL COLOR(D,D,C)
260 NEXT D
```

accomplishes three things (set D=2, then CALL COLOR(D,D,C), then increment D by one, then loop), the FOR loop in c99 does it all on one line. We say

```
for(d=1;d<=14;d++);
```

d is set to a, then tested to see if it is less than or equal to 14. The color(d,d,c) is executed as log as d<=14. As each color() function is executed, d is incremented by one by the "d++" statement. All things are done with one statement. Also remember that is there are multiple commands after a FOR statement in c99, they must be set off between a pair of braces. If a single statement, as we have here, they can be used without the braces.

[5] If you don't have version 2.0 of the Compiler and, thus, can't use "FOR" loops, you can try this: use a "while()" function. For example, instead of

```
for(d=1;d<=14;d++)
   color(d,d,c);
```

use this:

```
d=1;           /* Step 1 in a loop: set d=1 */
while(d<=14)   /* Step 2 : test for d<=14 */
     {
       color(d,d,c);
       d++;    /* Step 3 : increment d by 1 */
     }
```

It will accomplish the same thing. This is only needed if you have version 1.32 on NOT version 2.0.

c99 Tutorial 1

by Warren Agee
Compuserve ID 70277,2063

This is my *first* utility word for C. I am NOT an experienced C programmer...I have had 2 days experience with C. So, this may not be the best way to do it, but it DOES work!!

This file contains the C source code for the definition of a new function, seg(), and a test program to demonstrate its use. seg() corresponds roughly to SEG$ in BASIC. It will take a chunk of one string and place it in another string variable. Both strings must be variables. You provide the variable which contains the string to take apart, the variable where you want the new string, and the starting and ending positions of where you want the chunk taken out. If str1 contains "APPLE PIE" and you wanted str2 to contain "APPLE", simply use: seg(str1,str2,0,4). Everything starts with zero, not one. So the first character is 0, the second is one, etc. seg() returns the new chunk in str2. str2 should be an "empty" variable. This may not make sense yet, but I have commented this listing thoroughly.

Run the compiler on this program, then assemble it, then run it (option 3 of E/A). Load the assembled program first, then the CSUP file which resides on the c99 disk. Program name is then START. Not exciting, BUT IT WORKS!!

```
/* C TEST PROGRAM            */
/* Warren Agee 10/26/85      */
/* written with c99, by      */
/* Clint Pulley              */
/* 38 Townsend Ave.          */
/* Burlington, Ontario       */
/* Canada L7T 1Y6            */
/* Freeware: $20 donation requested */
/* Test of the seg() function */

#include dsk1.conio
int p1,p2,c;              /* integers */
char str1[81],str2[81];  /* strings, 81 chars long */

main()
{
  p1=0; p2=4;                    /* take a segment of the string  */
                                 /* from position 0 to position 4 */
  c=putchar(12);                 /* clear screen    */
  locate(3,1);
  puts("Please enter string:\n");
  c=gets(str1);                  /* input string into str1 */
  seg(str1,str2,p1,p2);          /* NEW FUNCTION!          */
  puts("The new string is:\n\n");
  puts(str2);                    /* str2 holds the new, segmented string */
}
```

```
/* Function to segment a string (SEG$ in BASIC)     */
/* seg(str1,str2,char1,char2)                       */
/* str1=string to take apart                        */
/* str2=segment of str1 that is returned            */
/* p1,2  =   beggining & ending position of string */
/* positions start at zero!!!                       */
/* after call seg(), the new string is contained    */
/* in str2; the original string is not altered.     */

seg(str1,str2,p1,p2)        /* start of function */
int p1,p2;                  /* tells the compiler what p1,p2,str1,str2 are */
char str1[81],str2[81];     /* these were DEFINED in the main program, but */
                            /* you have to DECLARE them again, here.       */
{
  int index,lim;            /* These are variables internal to the function;  */
  index=0;                  /* They do not relate to anything outside of this */
  lim=p2-p1;                /* function. ie. they are not global.             */

  while(index <= lim)
  {
   str2[index++]=str1[p1++];
  }

  str2[++index]=NULL;

}
```

Some Easy Learning
"C" Tutorials by Warren Agee
and more.

Suppose that we want to pass one or more values to a function. Look at this:

```
add(n1,n2)
int n1,n2;
{
  int sum;
  sum=n1+n2;
  return(sum);
}
```

The first line tells the compiler to expect 2 values in the parenthesis when this function is called. We give these two values the names n1 and n2. When one calls this function, two numbers may appear in parentheses [like add(1,2)] or two variables [like add(a,b)]. The next line is a variable declaration, which was described in the first tutorial, but the purpose here is a little different. The function add() receives two values; now the compiler has to know what KIND (class) of values they are. Since we are passing numbers, we declare them as integers. also notice that this must come *before* the opening brace. We then declare another variable, sum, to hold the sum of the two integers. We perform the addition just as one would do in BASIC. The next line is very important.

When this function is called, we give it two numbers, and we want back the sum, right? Since the variable "sum" is local to add(), once we return to the calling program, the value of sum is lost. "Sum" only exists in add() and nowhere else. What we have to do is artificially send the value of "sum" back to the calling program, and we do this with the return statement, as shown above. Now, when we call add(), we will get back the value of sum, like this:

```
main()
{
  int c;
  c=add(5,2);
}
```

The expression "add(5,2)" is replaced by the answer, and we assign that value to c. If we just wrote "add(5,2)" and did not assign it to anything, the sum would just be discarded.

But why do all this? We could just declare "sum" as an external variable in main(). That way "sum" would retain it's value throughout the entire program. In very large programs, you can run into difficulties if you use only external variables. Stick to local (automatic) variables whenever possible.

Well, there you have it! There is a lot more to cover as far as functions go. The return statement only returns ONE value, no more. If you need more than one value back from the function, you have to use pointers. Pointers can be quite sticky and confusing to beginners, so I will be spending quite some time on them in the next few tutorials. So stay tuned, and experiment! It's the only way to learn! (Well, reading my tutorials may help a bit!)

c99 Tutorial 2
"How To Function Properly"

By Warren Agee
Compuserve ID 70277,2063

In my first tutorial, I covered storage classes, something necessary to know before you even start programming in C. Functions are another basic concept which must be grasped before writing C programs. Simply put, a function is a subroutine designed to perform a specified task. In some cases, values are passed to and from functions, while other functions require no communication. Numerous functions are part of the standard C library, like gets() and puts(), which allows input and output of strings, respectively. Others, like fopen(), are kept in function libraries and stored on disk. And,of course, you may write your own functions. Indeed, the process of writing a C program involves writing user-defined functions, then putting all these functions together into a runnable program.

So, where do we begin? First of all, naming conventions. Although a function may have a name of any length, the c99 compiler only recognizes the first six characters, and they may be only alphabetic. Unlike most other compilers, the underscore (_) is not allowed. Secondly, what distinguishes a function name from a variable is the presence of parentheses. Depending on the purpose of the function, the parentheses may be empty, like getchar(). If the function requires values to be passed to it, these are placed inside the parentheses, as in puts("\nHello there!") . Now that the cosmetics are out of the way, let's get down to creating a function.

As I mentioned in the last tutorial, to call a function, merely type in its name, followed by a semicolon. To alert the compiler that you are creating a function, omit the semicolon.

```
 clr()
{
  int c;
  putchar(12);
}
```

Here we define function called clr(). Note the missing semicolon. Also note that since the parentheses are empty, we are not going to communicate any values to the function. Next we have an opening brace, which signals the beginning of the function body. Note that the brace aligns with the first letter in the function name above; this is a standard C convention to make programs easier to read. Then we indent a few spaces, another convention. We then define the integer variable "c." Because this statement occurs inside the function body, it is local to that function (See Tutorial #1 for more info). The next statement is a standard console i/o function which prints a character to the screen whose ASCII value is in parentheses. In this case, putchar(12) simply clears the screen. We then find a closing brace which ends the function. Notice that the two braces line up.

c99 Tutorial 3
"How To Create a Function Library in c99"

By Warren Agee
Compuserve ID 70277,2063

Function libraries are simply collections of tested functions (or subroutines) which reside in separate files from the main program. This helps the programmer to avoid reinventing the wheel each time he writes a program. There are basically two code. The difference is that with source code the compiler has to process the code every single time you compile, while an object-code library is only compiled once.

Creating a function library using source code is the easiest of the two methods. Say you create a function strlen() which returns the length of a string. You could just type in the function's definition each time you need it, but a simpler way is to save the source code for the function in a separate file. If the strlen function is ever needed in a program, merely insert the following line at the start of your code:

```
#include "dskn.xxxx"
```

where n is the drive no. of where the file sits, and xxxx is the name of the file which contains the source code. The compiler will load in and compile the source code as if it were typed directly into the main program. The #include command works just like .IF (include file) of TI WRITER.

Creating a function library using object code is a bit more involved. You start out the same as before, with the source code of the function in question as a separate file. But, as in the case with strlen(), you also need the following three lines at the beginning of the file:

```
#asm
    DEF STRLEN
#endasm
```

The actual definition for strlen() would follow these lines. The first line tells the compiler that the following code is not in C but in assembler. The second line tells the computer to make the STRLEN code available to another program. Even though it is defined in this program, a totally separate program (main) will also have access to it. Note 1) the leading space before DEF (that is important) and 2) the function name is in capital letters. The third line tells the compiler that the assembler code ends and C code begins again.

The DEF directive can be used to externally define many, many functions at once; just separate each function name with a comma.

Now compile and assemble your "mini-file" which contains just one function. You now have a standalone function library consisting of the strlen() function that can be used in ANY program. But how do you go about linking it to your main program?

The next thing to do is add three more lines to the start of your main program:

```
#asm
   REF STRLEN
#endasm
```

Looks familiar! But instead of defining an external function, we are REFerencing one. This tells the computer that even though the main program will use the function STRLEN, it must look OUTSIDE the current program for its definition. Please note that you can REFerence more than one function as with the DEF directive. If you look at the STDIO file on the c99 disk, you'll note that it contains mostly REF's!

When your program is compiled and assembled, be sure to load in the STRLEN file that you already compiled before you run your program. Under E/A option 3, first load your main program, then CSUP, then any other required files, then your STRLEN file. Now you're all set to go!

The theory behind this is not that hard to grasp: instead of including the definition of strlen() within the main program, we compiled it separately as a standalone module. But without the REFs and DEFs, there would be no communication between the program module and the strlen() module. This momentary slip into assembly language allows us the opportunity to open a line of communication between separately compiled modules.

c99 Source Code - Tutorial 4
by Warren Agee
Compuserve ID 70277,2063

```
/* NEW FUNCTIONS: getint() and stoi()    */
/* The following is a short demo program */
/* demonstrating the use of getint() to  */
/* directly input an integer, and stoi() */
/* which converts a String TO Integer,   */
/* similar to atoi(); stoi returns a     */
/* status flag, which atoi() does not.   */
/* Various version of both functions     */
/* exist in the public domain, these     */
/* have been adapted for c99 by Warren   */
/* Agee.                                 */
/* To run: Compile the entire file, making */
/* sure CONV;C is in drive one.            */
/* When done compiling & assembling, Load  */
/* & Run first the object code of this file */
/* and then the CSUP file. Program name: START */
/* The demo routine may be deleted and      */
/* getint & stoi */
/* can be saved as a function library. Dont */
/* delete the #defines...they are needed    */
/* in both functions.                       */

/* getint() demo  */

#include dsk1.conv;c
#define STOP -1
#define NO 1
#define YES 0
#define EOF -1
main()
{
  int num,stat;
  char string[81];
  puts("This reads in integers until it detects\n");
  puts("a CTRL-Z.\n");
  while((stat=getint(&num))!=STOP)
    if(stat==YES) {
        itod(num,string,5);
        puts(string);
        puts(" is the number accepted.\n");
      }
    else
      puts("That was no integer...try again!!\n");
  puts("We're finished!\n");
}
```

```
/* getint()                         */
/* format:  status=getint(&num) */
/* status contains:                 */
/*    -1 : EOF was found            */
/*     1 : error (no #s)            */
/*     0 : successful input         */

getint(ptrint)
int *ptrint;
{
  char buffer[81];
  int index,ch;
  index=0;

  while((ch=getchar())=='\n'|ch==' ')
    ; /* do-nothing */
  while(ch!=EOF & ch!='\n' & ch!=' ' & index<81)
    {
    buffer[index++]=ch;
    ch=getchar();
    }
  buffer[index]=0;
  if(ch==EOF)
    return(STOP);
  else
    return(stoi(buffer,ptrint));
}
/* stoi(string,intptr) -   */
/* converts string to integer (intptr) */
/* and returns status report. */

stoi(string,intptr)
char string[];
int *intptr;
{
  int sign; sign=1;
  int index; index=0;

  if(string[index]=='-'|string[index]=='+') {

    if(string[index++]=='-')
      sign= -1;
    else
      sign= 1;
    }
  *intptr=0;
  while(string[index]>='0' & string[index]<='9')
    *intptr=10*(*intptr)+string[index++]-'0';
  if(string[index]==0)
    {
    *intptr=sign*(*intptr);
    return(YES);
      }
  else
    return(NO);
}
```

c99 Sourcecode - Tutorial 5
by Warren Agee
Compuserve ID 70277,2063

```
/* DRIVER for string routines */
/* This program expects the conv;c file from the c99 disk */
/* and the STRING.C file in drive 2.                      */
/* The STRING.C file should be renamed "string" as per    */
/* the #include directives below.                         */

#include dsk2.conv;c
#include dsk2.string
char bigstring[80],smallstring[80];
char answer[3];
main()
{
  int c,a;
  puts("Simple test of match and strlen\n\n");
  puts("Remember that all #s start at\n");
  puts("Zero!!\n\n");
  puts("Enter large (target) string:");
  c=gets(bigstring);
  puts("\n\nEnter small (search) string:");
  c=gets(smallstring);

  a=strlen(bigstring);
  itod(a,answer,3);
  puts("\nLength of first string is:");
  puts(answer);
  a=strlen(smallstring);
  itod(a,answer,3);
  puts("\nLength of second string is:");
  puts(answer);

  a=match(smallstring,bigstring);
  itod(a,answer,3);
  puts("\n\nThe match occurs at");
  puts("character #:");
  puts(answer);
}
```

c99 Tutorial 6
"Pointers" - Part I

By Warren Agee
Compuserve ID 70277,2063

Of all the aspects of the C language, pointers are the hardest for the beginner to grasp. However, once mastered, one will find that pointers are what makes C a powerful language.

Simply put, a pointer is an address, or memory location. When one declares a variable (like int c;), that variable resides somewhere in memory. A pointer to the variable "c" is the address where "c" lives. This is advantageous if we want to change a variable that is local to another function. Using pointers gives us a way to get through the barrier of being local to another function. Think of it as going through the basement to get the contents of a variable. So how do we do this?

```
int c;
int *ptr;
```

The first line just declares a normal int variable. The second line declares a *pointer* variable named "ptr." Pointer variables are preceeded with an asterisk. Now, the first line tells the compiler that we have an integer-type variable, and it's name is "c." The second line says that, first of all, we have a pointer variable. Its name is "ptr." In addition, ptr is going to point to an integer-type variable--that's what the purpose of the int in the second line. Right now, ptr does not point to anything at all. We have merely created a variable, and have told the compiler what kind of variable it will point to. Similarly, char *goose; declares a pointer variable called goose which will point to a char-type variable. Think of it this way: a pointer variable's purpose is to "look" at other variables. But you have to tell it what it is looking at...an integer or a char-type variable.

Now, if we want ptr to point to "c", we do this:

```
ptr=&c;
```

Notice that the asterisk is gone. The asterisk has two purposes, one of which is to DECLARE a pointer variable. The other purpose will come later. The "&" can be pronounced "the address of." So "&c" means the address of c. This statement assigns the address of c to ptr. If we now do c=5, what will ptr contain?? The same thing. ptr holds the location of the variable c. No matter what c contains, the location of c will not change. Variables cannot move around in memory. Ptr just contains a number, perhaps 15000, just a memory location. To tell ptr to look somewhere else, say the variable x, all you need do is ptr=&x.

Now is the time to make an important distinction:

```
int *ptr;     /* ptr is a pointer variable */

ptr=&c;       /* &c is a pointer constant  */
```

You can change the contents of a pointer variable. You cannot change a pointer constant--it is a number! Just like you can say x=3 but you cannot say 3=x. This may seem obvious, but this can get confusing later on. Just remember the difference between a pointer variable and a pointer constant. The first is a variable, the second is a number. A pointer variable contains a pointer constant, but you can use constants in other places as well. More on that some other time!!

Now that we know how to declare a pointer variable and assign it, what do we dowith it??. Well, look at the following:

```
 c=5;
 *ptr=5;
```

The first line is obvious; it assigns c the value of 5. But what does the second line do?? The same thing!! Here we are using a technique called "indirection," or, as I like to call it, going through the basement. ptr contains the address, or location of c. If you were to print the contents of ptr, you would have some large number. But once we put the asterisk in front of it, we are saying "look at ptr's address, and access what is sitting there." In this case, we are saying, "Ptr, you are looking at a variable. Put the 5 there." You are making two jumps at once...the compiler looks at the address in ptr, then jumps to that address and see what's there. Similarly, if we want to know the value at c, we can do this:

```
int d;
d=*ptr;
```

Get the address out of ptr, hop over, get the value sitting there, and assignit to d. We are accessing the variable c INDIRECTLY, by using it's address.

This seems like an awfully silly way to do things!! Why all this hanky-panky with pointers and go DIRECTLY to the variable in question? Look at this:

```
int *ptr;         /* declares an external pointer to an int */
main()
{
  int answer;     /* automatic (local) integer */
  ptr=&answer;    /* ptr now points to answer */
  add(5,2);       /* calls add() */
}
 add(n1,n2)       /* n1=5, n2=2 */
int n1,n2;        /* declares the above as integers */
{
  *ptr=n1+n2;
}
```

This itsy-bitsy program combines several things I have covered before. Take a good look at the pointer used. First of all, we only have one external variable here: ptr. If we were to move ptr inside main(), that would make it unavailable to add(). So we declare it as external. Then we declare answer to be an int. Now, using the address operator (&), assign the address of answer to ptr. Now that we have done this, we can access answer anywhere in the program.

Then we call the add() function. Once inside, we add the two numbers together, and, using the indirection operator (*), we tell the compiler, "Here is this sum. Go to the address contained in ptr, and deposit this sum there." When we exit this function and go back to main(), where does the sum end up? Why in answer, of course! Ptr contained the address of "answer." In fact, you can think of the compiler as a mailman. He looks inside ptr, gets the address, and delivers sum to the mailbox it found at that address...in this case, that mailbox is the variable "answer."

Note that in the above example, we used ptr to point to only one variable. Suppoed we want several answers, and we want to keep them in separate variables? All you need do is change the contents of ptr to point to whatever variable you want, like this:

```
ptr=&answer1;
(...)          /* calculate answer */

ptr=&answer2;

(...)

ptr=&answer3;

(...etc.)
```

Just by changing the contents of ptr, you can point to any variable you want.

The above examples are trivial. From the last article, you learned how to easily return a value back to the calling function using the return() statement. But return() only gives back one value. By using pointers, you can alter as many values as you want. For instance, supposed you want to swap the contents of two values. This would be done like this:

```
main()
{
   int x,y;
   x=2;
   y=19;

   switch(&x,&y);
}

switch(n1,n2)
int *n1,*n2;
{
   int temp;
   temp=*n1;
   *n1=*n2;
   *n2=temp;
}
```

X and y and local variables. Using normal means, we cannot change the values of x and y outside of main(). So, instead of giving add() just the 2 variable on a platter, we give them the addresses. In this way, add() can go through the basement and change the contents of x and y. So, in order to inform switch() that it is getting addresses (or pointer constants), we declare n1 and n2 to be pointer variables. Only pointer variables can contain addresses. n1 and n2 now hold the addresses of x & y. We create a "temp"orary variable, and we do the switch. Since n1 and n2 are pointer variables, to get at the actual values, we use indirection (*). If we had just n1=n2 instead of *n1=*n2, all we would be switching are addresses, but not the contents of the addresses. Just a pointer variable by itself holds an address. But with an asterisk, we access the value contained at that address.

The main thing to remember here is that you can pass values to functions easily. But in order to CHANGE the value of an outside variable, you must use pointers.

Wow!! Confusing, isn't it?! I suggest you reread this tutorial many times. Buy a book on C (a good one) and read all you can about pointers. I've tried to make things a bit clearer by using "ordinary" language (like "through the basement"). When fiddling with numbers and pointers, you will run into difficulty seeing your results because c99 does not have printf(), which allows the output of numbers. In our case, we must first convert the number into a string, then print out the string. This is done with the file called CONV;C on the release disk. Please refer to the file called CONVT.C in this DL for a little tutorial on how to use the CONV;C function to print out numbers. Next time, I'll cover char arrays and strings, and, eventually, the biggie, string arrays.

c99 Programmer's Reference Sheet
Compiled by Herman Geschwind
Compuserve ID 73557,3447

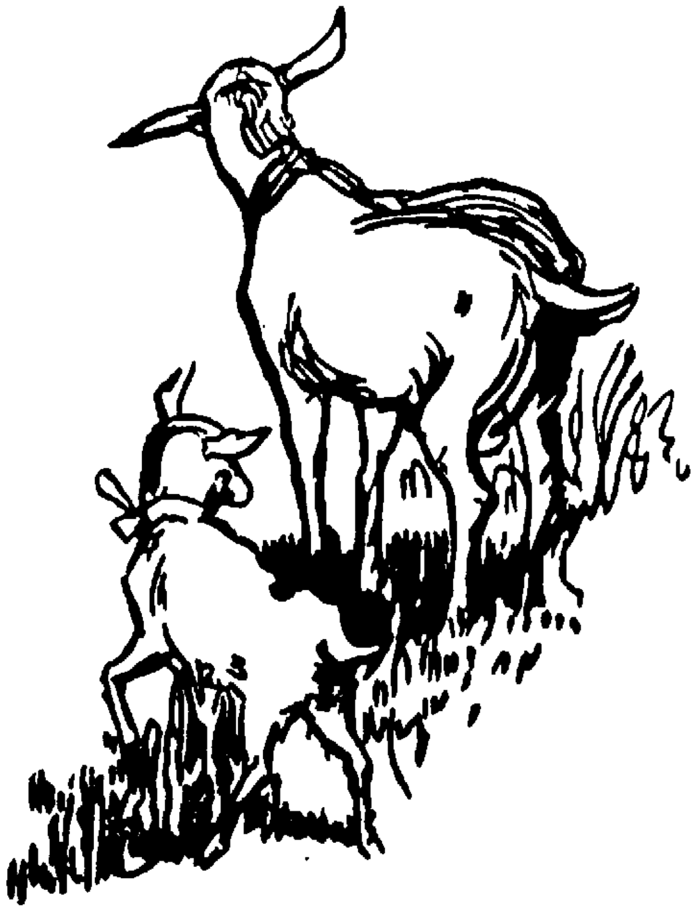
Command/Function	Description	Include File
c=getchar();	Read one character from the keyboard	CSUP
c-putchar(c);	Write one character to the screen	CSUP
c-gets(buff);	Read a line from the keyboard	CSUP
puts(string);	Write a string to the screen	CSUP
exit(c);	Exit the program	CSUP
abort(c);	Exit the program	CSUP
locate(row,col);	Locate the cursor on the screen	CSUP
key-poll(c);	Check keyboard status	CSUP
tscrn(f,b);	Change screen color	CSUP
unit-fopen(name,mode);	Open a file	stdio CFIO
c-fclose(unit);	Close a file	stdio CFIO
c-getc(unit);	Read one character from a file	stdio CFIO
c-putc(c,unit);	Write one character to a file	stdio CFIO
c-fgets(buff,col,unit);	Read a string from a file	stdio CFIO
c-fputs(string,unit);	Write a string to a file	stdio CFIO
c-fread(buff,len,unit);	Read a record from a file	Stdio CFIO
c-fwrite(buff,len,unit);	Write a record to a file	stdio CFIO
fseek(unit,recno);	Set record number	stdio CFIO
fdelete(filename);	Delete a file	stdio CFIO
c-feof(unit);	Test for end-of file	stdio CFIO
c-ferrc(unit);	Get error code	stdio CFIO
rewind(unit);	Rewind a file	stdio CFIO
grf1();	Set to graphics 1 mode	grf1rf GRF1
text();	Set to text mode	grf1rf GRF1

Command/Description	Function	Include File
screen(c);	Set screen color to c	grflrf GRF1
color(cs,f,b,);	Change colors for char set cs to f/b	grflrf GRF1
chrdef(ch,str);	Define character patterns	grflrf GRF1
chrset();	Load standard character patterns	grflrf GRF1
patcpy(a,b);	Copy character pattern	grflrf GRF1
clear();	Clear the screen	grflrf GRF1
hchar(r,c,ch,n);	Place character n times horizontally	grflrf GRF1
vchar(r,c,ch,n);	Place character n times vertically	grflrf GRF1
c-gchar(r,c);	Return value of character at r c	grflrf GRF1
s-joyst(u,&&x,&&y);	Read joystick u	grflrf GRF1
c-key(u,&&s);	Read keyboard u	grflrf GRF1
sprite(spn,ch,col,dr,dc)	Define sprite	grflrf GRF1
spdel(spn);	Delete sprite	grflrf GRF1
spdall();	Delete all sprites	grflrf GRF1
spcolr(spn,col);	Set sprite color	grflrf GRF1
sppat(spn,ch);	Set sprite pattern	grflrf GRF1
sploct(spn,dr,dc);	Set sprite location	grflrf GRF1
spmag(f);	Set sprite magnification	grflrf GRF1
spmotn(spn,rv,cv);	Set sprite velocity	grflrf GRF1
pmct(n);	Enable sprite automotion	grflrf GRF1
spposn(spn,&&rp,&&cp);	Return sprite position	grflrf GRF1
dsq-spdist(spn1,spn2);	Return distance between sprites	grflrf GRF1
dsq-spdrc(spn,dr,dc,);	Return dist. betw. sprite and loc.	grflrf GRF1
flg-spcnc(spn1,spn2,tol)	Sprite coincidence	grflrf GRF1
flg-spcrc(spn,dr,dc);	Coincidence sprite and location	grflrf GRF1
flg-spcall();	Coincidence of all sprites	grflrf GRF1
float number[FLOATLEN];	Define float type	floati FLOAT

Command/Description	Function	Include File
c-fpgets(s,f);	Prompt for floating point number	floati FLOAT
fpput(f,s);	Display floating point number	floati FLOAT
c-itof(i,f);	Converts integer to floating point	floati FLOAT
i-ftoi(f);	Converts floating point to integer	floati FLOAT
c-stof(s,f);	Converts string to floating point	floati FLOAT
c-ftos(f,s,mode,sig,dec)	Float array to string array	floati FLOAT
c-fexp(f1,op,f2,res);	Execute float expression	floati FLOAT
c-fexp(f1,"+",f2,res);	Add two numbers	floati FLOAT
c-fexp(f1,"-",f2,res);	Subtract two numbers	floati FLOAT
c-fexp(f1,"*",f2,res);	Multiply two numbers	floati FLOAT
c-fexp(f1,"/",f2,res);	Divide two numbers	floati FLOAT
true-fcom(f1,rel,f2)	Compare two floating point numbers	floati FLOAT
c-fint(f1,f2);	Returns greatest integer value	floati FLOAT
c-fcopy(f1,f2);	Copy one float array to another	floati FLOAT
filptr-topen(n,a,s);	Open a file(name,access,fsize)	tcioi TCIO
eof-tread(b,r,f,&&s);	Read a file(buff,rec,fileptr,&&size)	tcioi TCIO
eof-twrite(b,r,f,s);	Write a file(buff,rec,fileptr,size)	tcioi TCIO
eof-tclose(fileptr);	Close a file	tcioi TCIO
randomize();	Initialize random seed	random;c
rndnum();	Generate a 16-bit random number	random;c
rnd(n);	Generate a random number betw. 0&&n-1	random;c
n-atoi(s);	Convert string to integer	conv;c
s-itod(nbr,str,sz);	Convert number to signed decimal	conv;c
n-xtoi(hexstr,nbr);	Convert hexstring to integer	conv;c
bitmap(fore,back);	Change to bitmapped screen mode	biti BITSUP
bitclr();	Clears the entire screen	biti BITSUP
plot(x,y,c,t,);	Turns on single pixel	biti BITSUP

Command/Description	Function	Include File
line(xl,yl,x2,y2,c,t);	Draws line between two points	biti BITSUP
rect(xl,yl,x2,y2,c,t);	Draws a rectangle	biti BITSUP
circle(xc,yc,r,c,t);	Draws a circle	biti BITSUP
bitxt();	Copies ASCII characters into CPU RAM	biti BITSUP
bputch(ASCII,r,c,col);	Similar to putchar()	biti BITSUP
bputs(r,c,col,str);	Similar to puts()	biti BITSUP
blanks(r,c);	Places a blank on the screen	biti BITSUP
btblanks(r,c,count);	Blanks sequence of locations	biti BITSUP
bgetch(r,c,col);	Returns keypress of user input	biti BITSUP
bgets(buffadr,s,r,c,col)	Inserts characters in buffer	biti BITSUP
getky();	Scans keyboard similar to poll()	biti BITSUP

Notes: The purpose of "c99 Quick Reference" is to provide a handy summary of c99 command syntax and required parameters, a brief dscription and a reference to "include" and "object" files required to support a particular command. All references were re-capped from Clint Pulley's release diskette for c99 Version 2.0 except for "biti" and "bitsup" which are based on Jay Holovacs BITRTN and BITWRT Rel. 2.0. By necessity the description of the command had to be brief and is intended to be more of a "memory jogger". In all cases the user is urged to refer to the full documentation for all items .The naming of include and object files reflect the preference of the compiler of this quick reference. You may have your own system and can feel free to use any suitable editor to make necessary changes.



Going FORTH

Stepping FORTH into a new language with your 99/4A, and Geneve

by Howie Rosenberg
Compuserve ID 74216,1640

The FORTH language was developed by Charles Moore in 1969. As he stated, he developed the language as an interface between him and the computers he programmed. He placed the language in the public domain. The language has been promoted by the Forth Interest Group(FIG) of San Carlos California. FIG has available Assembly source code and architecture guides for each major processor for a nominal fee. These items are in the public domain. Both major versions of FORTH available for the TI-99/4A were derived from the FIG model.

In 1983 version 1 of Wycove FORTH became available. A short time later the TI version of FORTH was released to the public domain. There were flaws in both version. First were(are) a number of bugs which carried over from the FIG model. Several bugs peculiar to each of the versions also existed. The Wycove version had one fairly serious flaw in that method of storing data (screens) was somewhat flawed and the FORTH editor could not be used to full advantage. Proponents of the Wycove version claim increased speed which while true is considered not of any significance by most FORTH programmers as indicated by the fact that the TI version has gained much wider acceptance. Version 2 of Wycove FORTH while it offers some improvement of the screen structure, still was not the same as the FIG standard. There is still a debate in some quarters as to the relative merits of the two versions. I feel these are somewhat academic. TI IS the standard in our community and will most likely remain so. Whether it is due to the merits of the two versions or simply because the TI version was free is of academic interest.

ON STACKS, RPN, AND OTHER FORTH "HORRORS"

The characteristics of the language which are noticed, commented on, and in many cases used as an excuse to quickly depart for more traditional languages are all based on a simple idea one which is a central theme of Charles Moore's FORTH. Make it simple for the machine not necessarily for the programmer. This results in the highest degree of flexibility and speed in a higher level language. Thus while stacks are used internally in the architecture of all computers, not only are the stacks accessible in FORTH but must be utilized. The parameter stack is the only way to transfer data. The FORTH programmer enters data on the stack prior to executing a word. The resultant data from the word is outputted to the same parameter stack. In addition the return stack is readily available for use, indeed must be used in many applications so that the programmer must keep track of the status of this stack. This idea of putting numbers on the stack for use of the next word leads to the statement by many that FORTH uses Reverse Polish notation(RPN). Thus instead of 1+1=2 we have 1 1 + . 2 in FORTH. It is actually somewhat ironic in the TI world. For a long time, prior to the TI99-4 computer a long time competition existed between the two giants in the calculator world, TI and Hewlett Packard. Texas Instruments calculators all utilized an algebraic system AOS which TI claimed simulated the way people did arithmetic. On advanced calculators up to 9 levels of parentheses were allowed and arithmetic expressions were(and still are evaluated by entering equations left to right, with parentheses used as needed to indicate deviations from the normal hierarchy(first exponentiation followed by multiplication/division and finally addition and subtraction). The Hewlett Packard calculators used RPN and the user had to chew his way out from the middle of an expression and understand what he was doing to a much greater extent than did the TI calculator user. TI calculators were easier to use without much training or thought. Hewlett Packard calculators ran faster and, when comparing programmable calculators were considerably more efficient in terms of programming space. Based on calculator history RPN in a TI machine is indeed ironic. Another factor which seems to keep some programmers away from FORTH is the fact that the primary arithmetic system for FORTH is fixed point rather than floating point. Numbers can be single length(2 Bytes), double length(4 Bytes) or if needed the programmer can define even larger numbers. The use of fixed point arithmetic leads to efficient and fast running code. Sacrificed is ease of use. The programmer must understand any arithmetic manipulations used in his programs, size the results, decide on accuracy versus range of answers and the like. In short easy for the machine, a bit more difficult for the programer. Of course in both TI versions floating point routines are provided. Actually the floating point routines are links to the console GPL routines with there inherent lack of speed. There are cases where floating point is quite useful. Some FORTH systems have included hardware floating point which not only does not slow down the language but can run faster than software fixed point routines. In summary the use of the stack, RPN, and fixed point arithmetic as used in the FORTH environment is quite natural, leads to efficiency and speed in a higher level environment and really is well worth the effort for those who are willing to make the effort to learn how to deal with them.

WHAT IS FORTH?

FORTH IS A THREADED INTERPRETIVE LANGUAGE. The use of "interpretive" in this instance is somewhat confusing as the run time code is actually compiled code. FORTH applications consist of "words". New words are defined which call on previously defined words not unlike the concept of procedures in LOGO. Those words which are included in the basic FORTH language i.e. the primitives are called the kernel. The words in the kernal and any new words added in a particular application comprise the FORTH dictionary. Any new application has all words from previous applications which are presently in the dictionary available to it.

FORTH IS AN OPERATING SYSTEM. Moore's basic aim in designing FORTH was to provide an operating environment which while operating a higher level language would provide the maximum efficiency and speed at run time. To this end the FORTH system was designed. The system provides a disk operating system which was foreign to TIers and which still causes difficulty to many. A FORTH disk is divided into screens. Each screen consists of 16 lines of 64 Bytes of source code, Text, data, or program image. Each screen thus requires 1024 Bytes or 4 sectors. In TI FORTH after the FORTH system is booted, screen #3 is automatically loaded thus enabling auto start of an application or customizing the configuration. Five screen buffers are provided. These are used to store screen information on command. When all five buffers are full, a subsequent request for screen data results in the screen which was accessed least recently to be reused. Thus the FORTH disk system is a virtual memory. The utmost in simplicity and flexibility are provided in the operating system which allows for easy alteration. Many functions can be altered merely by changing the value of a user variable.

FORTH IS AN ASSEMBLY LANGUAGE. There is an assembler built into FORTH and words can be defined directly in assembly language as well as in higher level FORTH. The end result is similar to that which many of our EXTENDED BASIC programmers have been doing namely using the higher level language to provide simple non time critical functions and linking to assembly routines where needed. The process is somewhat simplified in FORTH as the code routines are direct replacements for higher level FORTH words. The process of linking is automatic. There are versions of FORTH not available for the TI-99/4A which have the ability for direct compilation of runable object code which can be run in the system without booting FORTH (i.e. establishing the FORTH environment). The result of such a compiler is Assembly object code. Supposedly all Atari arcade games which were produced for various machines including the TI-99/4A were written in FORTH and processed with a target compiler.

FORTH IS EXTENSIBLE. Changes can be rather easily made to any words in the dictionary. Of course care must be used when changing words in the kernel which are used by other words or the system will most assuredly crash. I can think of no other language which can be changed with such ease.

THE FORTH ARCHITECTURE

Maximum utilization of the FORTH language requires some understanding of the architecture of the language. This is more true of FORTH than other languages in that the elements of the language, stacks, users tables etc. are readily accessible to the user. For purposes of this note a short description is sufficient. TI FORTH utilizes memory much like the typical FIG FORTH system. Lower memory is used for support functions, the disk buffers, and the return stack. Upper memory contains the dictionary at one end, and the terminal input buffer at the other end followed by the parameter stack. The stack and dictionary are thus able to grow toward each other. Applications which require a large number of stack entries(unusual) can thus be handled by keeping the dictionary small. In turn by keeping the stack small, large applications can be handled.

THE STATUS OF TI FORTH IN OUR COMMUNITY

FORTH has been with the TI-99/4A community for 3 years. The FORTH programming community is not large but with few exceptions once a programmer has taken the trouble to learn FORTH and has started to use it he stays with it. There have been few commercial FORTH programs but those,which are available illustrate the capabilities of the language quite well. There is also a considerable array of public domain software for the TI written if FORTH.

Within the FORTH community there has been several major versions of the language after the FIG version. The latest of these is FORTH 83. While FORTH 83 has features which cannot be utilized in the 99/4A environment because of memory restrictions, the language is, generally transportable. Of course as always machine specifics in any language act as a restriction to transportability. Those TIers who try their hand at programming other machines will find that FORTH programming experience on he TI will be entirely applicable. Those of us who stay with the TI have found a language which has given us much greater control of your programming environment than available with other languages.

Introduction to FORTH .

(As lectured by Chick De Marti)

- INTRODUCTION -

FORTH is all things to all people. It is extensive (you can do anything in FORTH). It is fast (almost as fast as ASSEMBLY). It is EASY (to the extent it is user friendly) and it is complex (it can challenge the mind of the ASSEMBLY Programer).

While many routines appear to be simular to BASIC or EXTENDED BASIC, (see PLATE 1) these languages can not be compared to each other. FORTH, like FORTRAN, COBAL etc. is concidered to be a 'HIGH LEVEL' language. While it uses words that are common in the English language, it requires less interpretation into machine language than most of the other languages.

Because of it's structure, FORTH uses no "...run-time error checking. FORTH's compiled code is compact ...(it's) applications require less memory than their equivalent ASSEMBLY' programs!" (1)

FORTH is transportable (it has been used on just about every mini- and microcomputer known to the industry). Charles Moore who invente FORTH in 1969 said in all computer languages we, the operaters, have to learn the computer's language. He created FORTH, a language with whic we are able to teach the computer only those words required to complet an assignment.

IS FORTH A GOOD LANGUAGE?

"...First, FORTH is more than just a language. It can be a stand-alone operating system that provides basic support for terminal and disk control.

"Multi-tasking and multi-user FORTH systems are available. FORTH has been called a psuedo-machine language because the key words used for moving data from place to place are simular to the techniques used in assembly language.

"FORTH is an on-line interpreter. Commands are given to FORTH from the keyboard in a manner simular to the 'immediate mode' of most Basic interpreters. This is ideal for the development and debugging of the program. The programer can try out sequences of commands, one at a time. After the programer is satisfied that the sequence works properly, he can make it a permanent part of FORTH by giving it a name. Later, i can be called (type it's name) to perform by itself or as part of an other defined word.(1)

FORTH was first used as a computer control for large telescopes. While it continues to be used by many observatories, it also is being used to control ROBOT cameers, remote sensors of water depth and as an aid in navigation of large barges in inland waterways. General Electri also uses it to diagnose and trouble shoot large electric locomotive and it has been used in weather prediction programs.

To program in FORTH, you must know what a STACK is because almost all FORTH operations involves a STACK in some way. When adding 2 + 2, both numbers must be on the STACK and the sum is placed on the top of the STACK. The same goes for subtraction or multiplication or any operation. The STACK is actually the MEMORY AREA

You will learn to understand the function and operation of the stack both from outside and within a loop. Also, you will learn to store information and move it at will. With 'hands on' experimenting, you will become comfortable in FORTH and with your new found confidence, you will be able to let your own imagination dictate the programs you can write. The least you should accomplish is to be able to confidently enter and run the various programs that will appear (and are appearing) on the Source Boards, in books, magazines and Computer Group Newsletters.

SUGGESTED READING

There are many magazines and books dedicated to the furthering of our education in FORTH. MICRO (magazine) continues to increase it's articles on FORTH. Another excellent source of information is FORTH DIMENTIONS. MILLER GRAPHICS puts out an excellent Newsletter...and for the more ambitious programers, FIG (FORTH INTEREST GROUP ..PO BOX 1105, San Carlos Calif. 94070) publishes a bi-monthly newsletter. Membership in FIG is $15.00. Other suggested reading is:

STARTING FORTH by Leo Brodie
    published by Prentis Hall
THE FORTH MANUEL (of your choice)
INVITATION TO FORTH by Katzan
    published by Petrocelli Book
FORTH PROGRAMMING by Leo J. Scanlon
    published by Howard W. Sams Co.

VARIETIES OF FORTH

The main standards of FORTH that exist are FIG/FORTH, FORTH 79 and FORTH 83 (which is an update of FORTH 79). Some spinoffs are WYCOVE FORTH and TI-FORTH (an extension of FIG-FORTH). All are outgrowths of the original FORTH Inc. started by Charles Moore.

FORTH is extensible. It's programs are interchangeable with most othe computers. Included are APPLE, IBM and the VIC family (20 and and the 64)..as well as TEXAS INSTRUMENT'S 99/4A. The resident words that one computer may contain can easily be defined in another languag. An example ... Apple's 'HOME' can be defined : HOME cls 0 0 GOTOXY ;

Many of the differences have been documented in both Brodie's STARTING FORTH and Leo Scanlon's PROGRAMMING IN FORTH.

FORTH'S STRUCTURE

RESIDENT AND OPTIONAL WORDS

" The ACT of programming in FORTH is the act of defining 'WORDS'" WORDS can be made up of other user defined words..." and continue until a single word becomes the application desired. (2)"
Each new WORD is added to the dictionary and can be used in the definition of future programs. The format of a WORD is:

```
: name operation ( or data section ) ;
```

The colon at the beginning tells the compiler that the following items are the components of a 'WORD'. The 'NAME' can be of any combonation of letters and numbers, ie ?NOTE -P13 MOV/B etc. (also see CLASS 2). The 'DATA' can be a CONSTANT, A VARIABLE, LIST OF VARIABLES or TEXT. The semicolon denotes the end of the WORD definition.

Since a FORTH word must exist before it can be referenced, a bottoms up programming decipline is enforced" (2) Thus we must learn to program "... from the bottom up" (2). Words take their parameters from the 'STACK' and place the results on the STACK

AREAS WE WILL COVER

Besides 'RESIDENT' words you have a choice of 20 optional or 'ELECTIVE BLOCKS' you may add to the computer's memory. We will work primerily with <S> -SYNONYMS, <E> -EDITOR, <V> -VDPMODES and <P> -PRINT.
(See Page 5 Chap.1 of TI FORTH Manuel for a complete list.)

```
* NOTE *   <S> includes -DUMP -TRACE -COPY
           <V> includes -TEXT -GRAPH1 -MULTI -GRAPH2
           <P> includes -FILE
           <E> includes -64SUPPORT
```

STACK MANIPULATION WORDS:

```
DUP     ROT
DROP    -DUP
SWAP    >R        ( R> )
OVER    R
```

ARITHMATIC OPERATORS:

```
+  -  /  *  and later  MOD      AND
                       /MOD     OR
                       */MOD
```

REQUIRED EQUIPTMENT

```
COMSOLE                      EDITOR/ASSEMBLER MODULE
MONITOR                      RS232 INTERFACE (optional)
MEMORY EXPANSION             PRINTER         (optional)
DISK DRIVE (For the time being we will be referring
            to one drive - your drive #1 (actually
            Drive 0, but more on this later.)
```

STARTING UP YOUR SYSTEM:

```
1 ... Put your "SYSTEMS" disk in the drive.
2 ... Turn on EDITOR/ASSEMBLER Module.
      (use OPTION 3 ... LOAD)
3 ... type  DSK1.FORTH              <ENTER>
4 ... type  -EDITOR  -SYNONYMS  -PRINT

5 ... TO EDIT A SCREEN:
      (A) type 3  EDIT  <ENTER>  ( this gets you onto screen 3 )
      (B) use ARROW keys to move the cursor
      (C) press  FCTN  9  to get out of the EDIT mode.

(6) ... Take your SYSTEM disk out of the drive and replace it
        with a blank initialized disk (use your DISK MGR for
        the time being).  This will be your PROGRAM DISK.
7 ... type  EMPTY-BUFFERS           <ENTER>
8 ... type  (any number) EDIT       <ENTER>
      You will find that you have  90  blank screens on your
      program Disk.  Here is where you will store  your pro-
      grams and experiments.
```

CONGRADULATIONS!
You are now in FORTH and have executed 3 commands:

```
-EDITOR  -SYNONYMS  -PRINT  (all one group)
EMPTY-BUFFERS   and
(number)  EDIT
```

REMEMBER:

Only use your ORIGINAL FORTH disk to make a "SYSTEM" (or working) disk. A back-up copy can easily be made using the copier found elsewhere in this volume. At this point, let's try out some new words (commands):

UPDATE, LOAD and SWCH and UNSWCH

Entering FLUSH recopies the entire disk (like SAVE DSK1.xxx in BASIC). If you want to copy a particular screen from another disk, DO NOT FLUSH it to your disk...instead:

Type UPDATE <ENTER> This assures you that this particular screen is currently resident in your console. Put your disk in the drive and:

Type (n)LOAD <ENTER>

Where "n" is the number of the screen you want it copied to.

NOTE: A word of warning...ALWAYS EDIT screen(n) before you LOAD something to it...too many times we write over an important screen. If your planning on making changes to a certain screen, make a PRINTed copy of that screen BEFORE you change it. If your have already loaded the resident block of words under the title -PRINT (see #4 in apragraph "STARTING UP YOUR SYSTEM"), then you are ready.

Type SWCH (n) TRIAD CR UNSWCH

NOTE: This is very important... ALWAYS end your SWCH command with "UNSWCH". "SWCH" switches on your printer. If you do not include "UNSWCH" (unswitch), the printer will stay on...your console will become disables, as though it had crashed!

YOU DID IT! YOU DID IT!!

You now have some control of the FORTH environment...you can:

make a copy of an entire disk in FORTH	(FLUSH)
You can locate and examine a screen	(n EDIT)
You can print a copy of a screen	(SWCH n TRIAD UNSWCH)
You can copy a screen to a Prog. Disk	(UPDATE n LOAD)

And because you are getting used to the format, the language you are ready to peruse the volumns of misc. information put out by various books and newsletters. The following will be some I have selected as being worthwhile for the beginner. It does not represent all that is available, but you'll find it informative, instructive and interesting.

Go FORTH my friends. Chick

1. Booting The Forth System

   a. Insert the Editor/Assembler module.
   b. Switch on the P-box, monitor and console.
   c. Insert the SYSTEM DISK. If you have two drives, use #1.
   d. Press ENTER. Press 2. The E/A selections appear.
   e. Press 3. The file name request appears.
   f. Type, DSK1.FORTH (Press ENTER)
   g. The FORTH menue appears. Type, -EDITOR (Press ENTER)
   h. Type, -DUMP (Press ENTER)
   i. Type, 1 BLOCK DROP UPDATE (Press ENTER)
   j. Type, 4 BLOCK DROP UPDATE (Press ENTER)
   k. Type, 5 BLOCK DROP UPDATE (Press ENTER)
   l. Remove the SYSTEM DISK and relace it with a blank disk which will be formatted into a WORKING DISK.

2. Preparing the WORKING DISK

   a. Type, 0 FORMAT-DISK (The 0 is zero) (Press ENTER)
   b. Type, FLUSH (Press ENTER)

3. Entering a program on a SCREEN

   a. Type, 1 EDIT (Press ENTER). If the SCREEN is not clear, exit the SCREEN by pressing FCTN BACK .
   b. Type, 1 CLEAR (Press ENTER). This action clears the SCREEN but does not return you to the SCREEN.
   c. Type, ED@ (Press ENTER). This action returns you to the SCREEN in the EDITOR mode.
   d. The cursor is now on line 0, at the left margin. Type in the program listed on page 13 of "STARTING FORTH". On line 7, type the letter F . Do not use any punctuation marks. When program entry is completed, exit the SCREEN by pressing FCTN BACK .
   e. Type, 1 LOAD (Press ENTER). This action will load and execute the LETTER-F program.
   f. Type, FLUSH (Press ENTER), if you wish to save the program. This action writes the program to SCREEN #1 of the WORKING DISK.

<*<* FORTH and X-BASIC SIMULARITIES *>*>

	BASIC (or Extended)	FORTH	Secti Location
1.	" (to enclose a string)	." (needs an ending ")	RESIDENT
2.	: : (2 blank spaces	CR CR (carriage returns)	RESIDENT
3.	CALL CLEAR	CLS (also same on apple)	RESIDENT
4.	CALL CHAR(42,'1234'	0 1 2 3 CH	-GRAPH
5.	CALL COINC(#1,#2,8,C)	0 1 8 COINC	-GRAPH
6.	CALL COINC(ALL)	COINCALL	-GRAPH
7.	CALL COLOR(3,2,1)	0 1 2 COLOR	-GRAPH
8.	CALL COLOR(#1,12)	11 0 SPRCOL	-GRAPH
9.	CALL DELSPRITE(#1)	0 DELSPR	-GRAPH
10.	CALL DELSPRITE(ALL)	DELALL	-GRAPH
11.	CALL GCHAR(R,C,A)	C R GCHAR	-GRAPH
12.	CALL HCHAR(5,3,96,28)	2 4 28 96 HCHAR	-GRAPH
13.	CALL LOCATE(#2,80,120)	119 79 1 SPRPUT	-GRAPH
14.	CALL MAGNIFY(2)	1 MAGNIFY	-GRAPH
15.	CALL MOTION(#1,X,Y)	Y X 1 MOTION	-GRAPH
16.	CALL PEEK(-31880,A)	-31880 @	-GRAPH
17.	CALL PEEK(-31880,A):: PRINT A	-31880 ? or -31880 @ .	-GRAPH
18.	CALL POSITION(#1,Y,X)	0 SPGET	-GRAPH
19.	CALL SCREEN(7)	6 SCREEN	-GRAPH
20.	CALL SPRITE(#1,65,10,80,120)	119 79 9 65 1 SPRITE	-GRAPH
21.	CALL VCHAR(R,C,CH,COUNT)	C R COUNT CH VCHAR	-GRAP
22.	DISPLAY AT(12,18):ERASE ALL BEEP:"WE WANT FORTH"	11 17 GOTOXY CLS BEEP ." WE WANT FORTH "	

GOING FORTH

by David Aragon
512-826-8648
CompuServe ID 75766, 336

Most of you that have tried to learn FORTH have been directed to a book by Leo Brodie called "Starting FORTH." I must say that it is a very good book for the beginner. Mr. Brodie goes step by step throug the essentials of FORTH in a way that even a simple mind like mine car understand. There are, however, quite a few differences between his version of FORTH, FORTH-79, and the TI version. TI was nice enough tc put these differences into print for us, but somehow forgot to put the in any of their screens. I, therefore, have gone that one step farther. The screens listed below contains, I think, just about all c the changes to allow you to work through Brodie's book. It can be condenced so as to fit on a single screen that you could load prior tc working with Mr. Brodie's book. I might suggest that you add it to your menu as was discussed last month.

```
SCR #
   ( STARTING FORTH WORDS )
   : 2SWAP ROT >R ROT >R ;   : 2DUP OVER OVER ;
   : 2OVER SP@ 6 + @ SP@ 6 + @ ;   : 2DROP DROP DROP ;
   : 0> 0 > ;   : NOT 0= ;   : ?DUP -DUP ;
   : 2* DUP + ;   : 2/ 1 SRA ;   : 2! >R R ! R> 2+ ! ;
   : 2@ >R R 2+ @ R> @ ;   : NEGATE MINUS ;
   : I' R> R> R SWAP >R SWAP >R ;
   : U/MOD U/ ;   : D- DMINUS D+ ;   : DNEGATE DMINUS ;
   : DMAX 2OVER 2OVER D- SWAP DROP 0< IF 2SWAP ENDIF 2DROP ;
   : DMIN 2OVER 2OVER 2SWAP D- SWAP DROP 0< IF 2SWAP ENDIF 2DROP ;
   : 2CONSTANT <BUILDS , , DOES> 2@ ;
   : 2VARIABLE <BUILDS 0. , , DOES> ;
   -->

SCR #
   ( STARTING FORTH WORDS )
   : D= D- 0= SWAP 0= AND ;   : D0= 0. D= ;
   : D< D- SWAP DROP 0< ;   : M+ 0 D+ ;   : 'S SP@ ;
   : M/ M/ SWAP DROP ;   : >IN IN ;   : MOVE 2/ MOVE ;
   : DU< ROT SWAP OVER OVER U< IF DROP DROP DROP DROP 1 ELSE = IF
     U< ELSE DROP DROP 0 ENDIF ENDIF ;
   : TEXT PAD 72 BLANKS PAD HERE -
     1- DUP ALLOT MINUS SWAP WORD ALLOT ;
   : PLUS 32 WORD DROP NUMBER + ." =" . ;
   : ARRAY <BUILDS OVER , * ALLOT DOES> DUP @ ROT * + + 2+ ;
   : -TEXT 2DUP + SWAP DO DROP 2+ DUP 2- @ I @ - DUP
     IF DUP ABS / LEAVE THEN 2 +LOOP SWAP DROP ;
   : EXIT [COMPILE] ;S ; IMMEDIATE
```

Besides the resident WORDs in FORTH, you can create your own words. The formatof a FORTH word is:

```
     :  (name)    (instructions)    ;
```

The colon announces the start of a new WORD. The semicolon signals it's end. An example:

```
     :  BYE  EMPTY-BUFFERS  MON  ;
```

"BYE" will first clear the buffers of any memory (EMPTY-BUFFERS), then the word "MON" will take you bach yo the TI LOGO screen.

The following are J. VOLK's "most used words". Try 'em, you'll like 'em

EDitor

```
SCR #91
( MY MOST USED WORDS  by J. Volk)
( LOAD -SYNONYMS FIRST if not already BLOADed)
: MYLOAD -GRAPH -VDPMODES ; ( Will load these options )
: AT GOTOXY ; ( Same as 'Display At' )
: TOP CLS 0 0 AT ; ( Same as Brodie 's 'PAGE' )
: RANDOM RND 1+ . ; ( n RANDOM >>> gives random number )
: PICK ( Leave copy of n1-th number on top of stack )
       ( n1 --- n2 )
     2 * SP@ + @ ;
: ROLL ( Rotate nth number to top of stack ) ( n --- n)
     DUP 1 = IF DROP ELSE DUP 1 DO SWAP R> R> ROT >R >R >R LOOP
     1 DO R> R> R> ROT ROT >R >R SWAP LOOP THEN ;
: TEST BEGIN ." HELLO THERE" 2 SPACES ?TERMINAL UNTIL ; ( FCTN 4
 TO END )
: SGN DUP IF DUP 0< IF -1 ELSE 1 ENDIF ELSE 0 ENDIF ;
: WORK BLOCK DROP UPDATE ; ( My word to update a FORTH screen )

SCR #92
( A Word to copy FORTH disks-Single  Drive 5/16/84 J. Volk )
( Load Screen #91 and -COPY then RUN )
0 VARIABLE COPYSCR  0 DISK_LO !
: MES1 COPYSCR @ 88 > IF CLS ABORT ENDIF TOP 2 11 AT ." INSERT M
ASTER DISK          " KEY DROP ; ( PRINT MESSAGE AND KEY PRESS )
: COPY1 5 0 DO COPYSCR @ WORK 2 20 AT ." SCR # " COPYSCR ? 1 COP
YSCR +! LOOP ; ( DO THE WORK AND LET US KNOW-GET NEXT SCREEN )
: COPY2 2 11 AT ." INSERT COPY DISK-ANY KEY      " KEY DROP ;
( COPY 5 SCREENS AND PRINT MESSAGE )
: GETIT BEGIN MES1 COPY1 COPY2 FLUSH COPYSCR @ 89 = UNTIL ;
( RUNS ABOVE WORDS )
: MESO TOP 2 11 AT ." INITIALIZE FORTH DISK ? (Y/N) " ;
: MSG TOP 2 11 AT ." INSERT COPY DISK " KEY DROP ;
: RUN MESO KEY 89 = IF MSG 0 FORMAT-DISK DISK-HEAD ENDIF GETIT ;
 ( ROUTINE TO INITIALIZE DISK )
_
```

Forth Tutorial #1

By Warren Agee
Compuserve ID 70277,2063

PREFACE:

With this tutorial (and more to come!), I humbly submit what I have learned by programming in the FORTH language. One reason I decided to put down into words the knowledge I have acquired is to share my experiences, frustrations, and triumphs while hacking away with FORTH. But, on a more personal level, I give these tutorials to the TI world as a token of appreciation for everything I have gained from knowing such people as Ronald Albright, Barry Traver, and Howie Rosenberg, just to name a few, as well as the whole gang on the TI FORUM. These and many others have given unselfishly to both me and the TI community as a whole, and I am proud to be part of a community that refuses to die. Now, on with the programming, FORTHwith! <ugh!>

STRINGING ALONG IN FORTH

Of all the peculiarities the beginner confronts in FORTH, string handling is a major obstacle. Nothing is more frustrating than to sit down and have no idea how to write something like A$="1234"::A=VAL(A$). No advanced string-handling routines come with the TI FORTH systems disk. So, it is up to the programmer to invent his own. Hopefully, this article will make it much easier to write a FORTH program that involves any string manipulation at all.

THE BASICS

Before jumping into the new string words, let's first take a look at how a string sits in memory. This knowledge is imperative in order to fully exploit the power of FORTH. Think of a string as a numeric array; each character in the string represents a number, or byte. The string HOME COMPUTER would look like this:

```
--------------------------
|H|O|M|E| |C|O|M|P|U|T|E|R|
--------------------------
```

The first "box" represents the address in memory where this string starts. Determining the location of this address is what we will discuss next.

There are many ways to store strings; we could save them in VDP RAM, or in the disk buffers. In this article, we will investigate storing strings directly in the dictionary. A string variable is no more than a numeric variable stretched out. In fact, unlike BASIC, there is only one type of variable in FORTH. The only thing that differs is the size. First use the word VARIABLE to create a variable. But when you create it, let's say 0 VARIABLE TEST, only two bytes are allotted for storage. This is fine for single numbers; but for strings, we can use ALLOT to specify the length of the variable. For instance, 0 VARIABLE TEST 8 ALLOT will create a variable with a length of ten bytes. This gives us room for a string with a maximum length of 10 characters. If the above is executed, the variable will look like this in memory:

```
 --------------------
 | | | | | | | | | | |
 --------------------
 |
 |
 addr of TEST
```

Once the string is created in the dictionary, there may be garbage in the variable. Here we can use BLANKS to clean it out: TEST 10 BLANKS. This will fill ten bytes of memory, starting at TEST, with blanks (ASCII 32).

Now that space has been reserved for the string, there are basically two ways to store the string. If the contents of the variable is not going to change, then the word !" can be used. All this word requires is an address on the stack. So, to store STRINGS in the variable TEST defined above, the sequence TEXT !" STRINGS" will do the trick. If you wish the user to input the string, the word EXPECT is available, which is similar to BASIC's INPUT statement; it awaits an entry from the keyboard. EXPECT requires both an address and the maximum length of the string on the stack. Using TEST 7 EXPECT will achieve the same results as TEST !" STRINGS" . The variable will now look like this:

```
 --------------------
 |S|T|R|I|N|G|S| | | |
 --------------------
```

This presents our first problem. Since the contents of TEST is not expected to change, the length of the string can be assumed to always be 7. However, if the length will vary, we must keep track of it. EXPECT does not do this for us. Sure, it requires a length on the stack, but it does not incorporate this value into the string. Not to worry. This brings us to our first new word, ACCEPT, which replaces EXPECT. The only difference is that ACCEPT stores the actual length of the string inputted into the byte preceding the string. This is often called the count byte. If we use ACCEPT in the example above, our string would now look like this:

```
 --------------------
 |7|S|T|R|I|N|G|S| | |
 --------------------
  |
  |
  addr of TEST
```

As you can see, the first letter of the string, the "S", no longer sits at TEST; the whole string has moved over one byte to make room for the count. Now, to print this string is a trivial matter of using TEST COUNT TYPE. TEST supplies the addr of the complete string. COUNT takes that address, calculates the address of the actual string (TEST+1), and finally supplies the length of the string. Everything is ready for TYPE. To summarize what we have done so far, consider the following example:

```
0 VARIABLE COOKIE 18 ALLOT (reserves 20 bytes)
COOKIE 20 BLANKS
COOKIE 20 ACCEPT _CHOCOLATE CHIP_
COOKIE COUNT TYPE
```

Note: any words that appear between underscore characters (_) are to be typed in as a response to the ACCEPT word.

MOVING AROUND

Up till now, I have discussed performing basic functions on strings which reside directly in the dictionary. This is not always the ideal situation. A much better way is to store the string in a temporary spot, do what needs to be done, then move it back into the dictionary. This temporary spot is called PAD. Typing in PAD just leaves an address on the stack, just as TEST does. Typically, instead of typing in TEST 10 ACCEPT, you would type PAD 10 ACCEPT. Once any processing is done, the word CMOVE can move the bugger back to where it belongs. Here arises our second problem. CMOVE moves a specified quantity of bytes from low memory to high memory. But what if you want to go the other way around? Well, define a new word, of course!

The new word will be <CMOVE, which is included in some versions of FORTH. But wait--isn't it rather a hassle having to remember which word to use? Of course it is! Remember, FORTH is extensible, and we can make it as user-friendly as we like! The next new word will be CMOVE$, which decides which way the string is moving, and does the moving for you.

Here is an example of using CMOVE$ and PAD:

```
0 VARIABLE DRESSER 8 ALLOT
DRESSER 10 BLANKS
PAD 10 ACCEPT  _SOCKS_
.
. (string processing done here)
.
PAD COUNT                (get addr and length)
1+ SWAP 1- SWAP          (PAD-1 CNT+1)
DRESSER SWAP             (PAD-1 DRESSER CNT+1)
CMOVE$
DRESSER COUNT TYPE
```

Everything should make sense until you get to the 1+ SWAP 1- SWAP. The reasoning is a little hard to grasp at first: we want to move SOCKS from PAD to DRESSER. We also want to maintain that ever-important count byte. But when we use PAD COUNT, we only have the addr and length of the string itself, not including the count. So we compensate. Add 1 to the count (because we want to move the count byte along with the string), then subtract one from the address. COUNT adds 1 to the address, so we have to correct this to catch the count. Once these two numbers have been corrected to catch the count byte, shift things around to get everything ready for CMOVE$. To better illustrate this, here is a diagram of PAD:

```
 --------- . . .  . -------
 |5|S|O|C|K|S| | | | |      (Contents of PAD)
 ----------------------
  | |
  | PAD+1 (This is where you are using PAD COUNT)
  |
 PAD (This is where you are using PAD COUNT 1+ SWAP 1- SWAP)
```

If you can understand the principle of the count byte, and how to keep the count byte tacked on to the string when moved, then a major obstacle in writing in FORTH has been removed. Next time, I will discuss string arrays. Until then, experiment, and Keep On FORTHin'!

SUMMARY OF RESIDENT WORDS -

VARIABLE	(n--)	Create a variable.
ALLOT	(n--)	Reserves n bytes in the dictionary.
BLANKS	(addr n--)	Fills n bytes with blanks.
EXPECT	(addr n--)	Waits for input; stores string at addr.
COUNT	(addr--)	Returns addr and count of a string.
CMOVE	(adrl adr2 n)	Moves n bytes from adrl to adr2, from low to high memory.
PAD	(--adr)	Temporary storage place for strings.

NEW WORDS

```
: PICK  ( nl -- n2)

  2 * SP@ + @ ;

( Copies nlth number to top of stack)
******

: LEN   (addr -- n)

  255 0 ( string max=255 characters)
  DO
     DUP I + C@
     0= IF       ( looks for null)
       I LEAVE   ( I=length of string)
     ENDIF
  LOOP
  SWAP DROP ;

( Returns the length of a string at addr.)
```

```
: ACCEPT  ( addr n — )

  OVER 1+ DUP ROT    ( adr+1 )
  EXPECT
  LEN                ( length of string)
  SWAP C! ;          ( store count byte at addr )

( Waits for input; stores count at addr and string
  starting)
( at adr+1.)
******

: <CMOVE  ( adr1 adr2 n)

  DUP ROT + SWAP ROT
  1-DUP ROT +
  DO
    1- I C@ OVER C! -1
  +LOOP
  DROP ;

( Moves n bytes from adr1 to adr2, from high to low memory.)
******

: CMOVE$  (adr1 adr2 n)

OVER 4 PICK >
IF <CMOVE
ELSE CMOVE
ENDIF ;

( Moves n bytes from adr1 to adr2; automatically decides on)
( direction.)
```

Forth Tutorial 2
by Warren Agee
Compuserve ID 70277,2063

AN ARRAY OF STRINGS

Last time we met, I covered how to handle the basic string in FORTH. I also stressed the importance of the count byte and how to move it along with the string. Now, we have graduated to the realm of string arrays, which is an entirely new mess with which to work.

Think of a string array as a super-long string. Since the character (or bytes) of a string sit sequentially in memory, it stands to reason that the elements in a string array do also. But the physical structure of an array must be forced by the programmer; maintaining an array is not automatically done. The structure is what we will discuss first.

Here is a possible string array:

```
------------------------------------------
|3|C|A|T| |3|D|O|G| | | |4|B|I|R|D|5|P|O|O|C|H|
------------------------------------------
```

This array has 4 elements: CAT, DOG, BIRD, and POOCH. Fine, right? No way! This is a mess! Each element in this array has a different length. Element #1 has 5 bytes, #2 has 7, #3 has 5, and #4 has 6. How in the world are you going to keep track of all this? You cannot! Elements in a string array - must - have a constant length. A much better way to construct the above array is like this:

```
--------------------------------------------
#3|C|A|T| | #3|D|O|G| | #4|B|I|R|D| #5|P|O|O|C|H|
--------------------------------------------
```

Note: from now on, the boundaries between elements will be pound (#) signs. Now each element is exactly 6 bytes long. Remember, the actual strings in an array can have variable length, but each element has to have the same - maximum - length. If the string is shorter than the maximum, then blanks will fill the excess space.

So much for structure and theory. How do we go about achieving this neat and tidy array? Well, start out with good old' VARIABLE. Remember, arrays (string OR numeric) are just stretched-out variables. Think of a good name, let's say PETS. Now, decide how many elements this array is going to contain. Let's say 20. Now decide the maximum length of the elements. Let's keep it at 6. Remember to allow enough room for the count byte for each element! This sequence will then create our array:

```
0 VARIABLE PETS
60 ALLOT            (10 elements X 6 bytes each)
```

That's it! Easy, eh? Actually, you can think of the 60 ALLOT as a DIM statement in BASIC. It reserves memory for the array. The hard part is accessing the individual elements. Also notice that I totally ignored the initial two bytes which VARIABLE automatically reserves; when dealing with large arrays, the first two bytes are insignificant and may be ignored. This makes for much better readability when going over your program listings.

Now refer back to my diagram of the PETS array. The first box of the array is the address provided by PETS. Since the first element has a count byte, simply typing in PETS COUNT TYPE will print out "CAT". But how do you get at the rest of the array? You have to calculate the address of the element, using this simple formula:

```
base addr + (element # * length of each element)
```

The base addr is PETS. Now, as with most of FORTH, element numbers start at zero. Let's say you want the first element using this formula. Plug in the values: base addr=TEST, element #=0, length=6. 6 * 0 = 0, so you are adding 0 to the base addr to find the first element. That makes sense! Similarly, to get to the second element, the sequence to type in is (TEST 1 6 * +). What you are actually doing is adding an offset to the base address. Once you have the address of the element, a simple COUNT TYPE will print the contents, providing you stored the count byte! If you want to view all the elements in PETS, type in:

```
: GO 10 0 DO CR PETS I 6 * + COUNT TYPE LOOP ;
```

Since element #s start at zero, we want to print out elements 0-9. However, you must always add 1 to the upper limit whenever using DO LOOPs in FORTH.

As you can imagine, if you have a lot of string arrays, you will need to make these calculations often. To make it more readable (and more convenient), we can easily turn that into a definition, as follows:

```
: PETS() PETS SWAP 6 * + ;

: GO 10 0 DO CR I PETS() COUNT TYPE ;
```

This is MUCH easier to read than before. As a naming convention, I use the () symbol to indicate that PETS is an "indexing" word; all it requires on the stack is the index, or element #. A word of warning: When you are using DO LOOPS, the word "I" must used in the same definition as the loop itself. You cannot put the "I" in the definition of PETS(); it MUST appear in the same definition as the DO LOOP. This problem is actually a blessing in disguise. Since we removed the "I" from PETS(), we are free to use the index word outside of the loop. In other words, if all I needed was the last element of the array, I could just type in 9 PETS() COUNT TYPE. No loop is needed!

Up till now, all you have done is sit back with your arms folded and watch me babble on about accessing an array. Here's your chance to follow along with me as I show you how to store things in your array. First we will use ACCEPT and input the strings directly into the dictionary, then we will modify our routine so we first input into PAD. First of all, we have to modify our array a bit. In the above example, POOCH barely fit into the space allot for it--6 characters. If we are to use ACCEPT (which was defined in the previous tutorial) and input directly into the array, we need to tack on 2 more bytes for each element. You see, ACCEPT (and EXPECT) always glue 2 nulls onto the end of each string. So if you input a string exactly 6 characters long directly into PETS, ACCEPT will over-write the next element with nulls! With this in mind, here is the complete routine:

```
0 VARIABLE PETS
80 ALLOT            (10 items * (6+2) bytes each)
PETS 80 BLANKS
: PETS() PETS SWAP 8 * + ;
: INPUT-IT
  10 0 DO I PETS()  ( addr of each element)
  6 ACCEPT ( max. len for each string=6)
  LOOP ;
: PRINT-IT
  10 0 DO I PETS() COUNT TYPE
       LOOP ;
```

If you have been following since the first installment in this series, the mechanics of this loop are self-explanatory.

This is fine, but remember what I said about avoiding inputting directly into the array? To avoid those darn blanks from creeping in, Input the string into PAD first, then move them into the array. Here is the new routine: (remember to FORGET PETS first):

```
0 VARIABLE PETS 60 ALLOT   (10 items + 6 bytes)
PETS 60 BLANKS
: PETS() PETS SWAP 8 * + ;
: INPUT-IT
  10 0 DO PAD 6 ACCEPT
  PAD COUNT 1+ SWAP 1- SWAP
  I PETS()                 (Get addr of element #I)
  SWAP                     ( source addr,dest. addr, cnt)
  CMOVE$ LOOP              ( CMOVE$ was defined in the previous)
                           ( tutorial )
: PRINT-IT
  10 0 DO CR I PETS() COUNT TYPE LOOP ;
```

The PAD COUNT 1+ ... sequence seems confusing, but if you read my last tutorial, you should remember it. We want to move not only the string, but the count byte as well. But PAD COUNT returns the address of the string itself, along with its length. Subtracting 1 backs up the addr to the count byte; meanwhile, add 1 to the cnt on the stack so CMOVE$ will move the entire string+cnt. Also remember that I PETS() just returns the proper address of the element in the array; a similar sequence in BASIC would be:

```
100 FOR I=1 TO 10 :: INPUT PETS$(I) :: NEXT I.
```

Well, I've run out of room for this issue. Next time I will introduce some string array utility words which will allow you to do some heavy-duty string processing! Bye for now!

Forth Tutorial 3

by Warren Agee
Compuserve ID 70277,2063

Beyond the Basic String

In the past, we have looked at the basic string, how it sits in memory, and the basic string array, and how it sits in memory. We've learned how to store a string, retrieve it, and print it. Where do we go from here? Well, hopefully you have been playing around on your own with strings, along with some of the new words I presented (like ACCEPT). From now on, things are going get a bit more advanced, and the knowledge gained (hopefully!) from the first two tutorials is important. In this tutorial, I will be presenting some very useful and powerful string utilities that I have collected from countless sources; some of them I have written myself.

Some terminology, first: a BASE STRING is a string to which you want to do some sort of manipulating. A SUBSTRING is a separate string from the base string. You usually use it as a reference. For example, if we were to delete the word FOX from the sentence THE QUICK BROWN FOX, the sentence would be the base string, and FOX would be the substring. Also note that the utilities presented here work only with single strings and NOT string arrays. These words are INS$, DEL$, and -MATCH. First of all, let's say we reserve memory for a 100-byte long string called TEST$. We also have another string called SUB$. Here are the contents of these strings:

```
   ------------------------------------------------
   |22|N|O|W|  |I|S|  |T|I|M|E|  |F|O|R|  |D|I|N|N|E|R|
   ------------------------------------------------


   ---------
   |3|T|H|E|
   ---------
```

(You can use ACCEPT and type in the above if you want to follow along).

Notice that the first string is NOT an array, merely a long string which happens to be a sentence. The 22 is the count byte. Unfortunately, we seem to have a word missing! What to do? At the end of this tutorial is a definition for INS$, which will insert a "substring" into a "base" string. The stack arguments correspond as follows:

```
   INSERT$  ( adr1 n1 adr2 n2 adr3 --- )

   adr1 --> address of base string
   n1   --> length of base string
   adr2 --> address of substring
   n2   --> length of substring
   adr3 --> address of insertion point
```

So, using the above strings, assume that the word "THE" (the word that is missing) is located at SUB$. (Remember that variable names just supply an *address*, which is what we need for INSERT$ to work). Now to insert THE into the sentence, do the following:

```
TEST$ COUNT   ( adr1 n1)
SUB$ COUNT    ( adr2 n2)
TEST$ 9 +     ( point of insertion - addr3)
INSERT$
```

Your string will now look like this:

```
------------------------------------------------------------
|26|N|O|W| |I|S| |T|H|E| |T|I|M|E| |F|O|R| |D|I|N|N|E|R|
------------------------------------------------------------
```

Experiment with INS$ until you become comfortable with it; use the previously defined ACCEPT to store a long string at one location, and a substring to insert at another location. Just remember that YOU have to supply the location, or address, of the insertion point.

-MATCH

Now HERE is an interesting word! -MATCH looks for a matching string and returns a 1 if no string is found, and a zero (0) if it is found. Additionally, -MATCH also leave the address of the byte AFTER the match. It requires four stack arguments: the address of the base string and its length, and the address substring and its length. -MATCH tries to find an occurrence of the substring in the base string. This word is useful in conjunction with INS$ above. Here is one possibility using INS$ and -MATCH. Say you want to insert the word MY after the word FOR in the above string (TEST$). It might go something like this:

```
: GO
PAD 3 ACCEPT _THE_   (Word to search for)
                      Note: anything that appears between underscores (_) is
                      to be typed in as a response to ACCEPT.)
TEST$ COUNT          (Addr & cnt of base string)
PAD COUNT            (Addr & cnt of substring)
-MATCH               (stack: --- adr3 flag)
IF                   (1=no match)
  DROP ." Not found!"
ELSE                 (else found; adr3 is left on stack)
  CR ." ENTER NEW WORD:"
  PAD 10 ACCEPT _MY_  (Word to insert)
  TEST$ COUNT        (Addr & cnt of base string)
  PAD COUNT          (Addr & cnt of substring)
  5 ROLL             (Bring up adr3 which was left by -MATCH; this is the
                      insertion point)
  INSERT$
  CR CR TEST$ COUNT TYPE  (Displays new string)
ENDIF  ;
```

Please note that ROLL does not exist in the standard TI FORTH dictionary and must be defined separately. That definition appears at the end of this article.

DEL$

Finally, we come to DEL$, which, by no surprise, deletes a substring. It works along the same lines as INS$; the stack arguments require the address and length of the base string and the substring. DEL$ searches the base string, looking for a match with the substring. It accomplishes this by using -MATCH, explained above. Once it finds a match, it deletes the string. If no match is found, it clears the stack and exits, no harm done. If you plan to use DEL$ in a program, you may want to modify it a bit. With -MATCH, you can test to see if a match is found. Perhaps you want to do the same with DEL$. You could very easily leave a 1 on the stack if the string was found and deleted, or leave a zero if no match was found. Examine the comments for the listing of DEL$ to demonstrate how to do this.

Well, that's it folks! FORTH is a powerful language, but it lacks in some areas, especially string handling. But the real power in FORTH lies in its extensibility. As demonstrated here, we now have a good number of basic string utilities which can now become part of our FORTH vocabulary of words. Does XBASIC have a built-in INSERT or DELETE function for strings? Sure, you can simulate it with SEG$, but that is very clumsy and VERY slow. With a little bit of ingenuity, you can make FORTH run circles around most languages without sacrificing ease-of-use. Till next time, have fun!!

<u>DEFINITIONS OF NEW WORDS</u>

```
: ROLL DUP 1 = IF DROP ELSE DUP 1 DO SWAP R> R> ROT >R >R >R LOOP 1
  DO R> R> R> ROT ROT >R >R SWAP LOOP ENDIF ;
```

(NOTE: the following definitions require the word PICK which was defined in an earlier article in this series.)

```
: INSERT$  (adr1 n1 adr2 n2 adr3 ---)
  DUP 6 PICK 6 PICK +
  1+ OVER -
  OVER 5 PICK + SWAP <CMOVE
  OVER 5 ROLL + 5 ROLL
  1- C! SWAP <CMOVE ;

: -MATCH   (adr1 n1 adr2 n2 -- adr3 flag)
  SWAP DUP C@ 5 PICK 5 ROLL +
  DUP 1 SWAP 6 PICK - 1+ 7 ROLL
  DO
    3 PICK I C@ =
    IF
      0
      6 PICK 1
      DO
        J I + C@ 6 PICK I + C@
        = NOT
        IF
          DROP 1 LEAVE
        ENDIF
      LOOP
      IF ELSE
        DROP DROP I 4 PICK + 0
```

```
          LEAVE
        ENDIF
      ENDIF
    LOOP
    ROT DROP ROT DROP ROT DROP ;

: DEL$    ( adr1 n1 adr2 n2)
  4 PICK 4 PICK


  4 ROLL 4 PICK -MATCH
  IF
    DROP DROP DROP DROP      (NOT FOUND)
                             (clear stack)
    ( 0 )                    (insert the 0 if you want to leave a flag if not
  ELSE                        when not found)
    DUP 3 PICK -
    5 PICK 5 PICK +
    3 PICK - 1+ CMOVE
    - SWAP 1 - C!
    ( 1 )
                             (insert the 1 if you want to leave a flag if match
  ENDIF ;                     was found)
```

SOME REFRESHMENTS

SOME NEW SOFTWARE . . .

UCSD Pascal
Logo
Pilot

Some New Software

Things They Don't Tell You About The P-System
by Jerry Coffey

I put my first P-system together about a year after I bought my TI99/4A console for $49.95. In the intervening year I had acquired an expansion box, 32K memory, and a "disk memory system". I watched the UCSD (actually Softech) software prices drop but found the P-code peripheral card disappearing from the shelves even faster. Finally I gritted my teeth and bought the disks before I found a card to run them. In desperation I contacted a TI repair center and talked them into selling me a card outright. It was then I discovered how primitive my single drive system really was. I had to have another drive or give up the whole system as an expensive mistake. In the the years since, I have bought and sold a lot of other hardware and with each up upgrade I have learned something new about the P-system — both the quirks of third-party hardware and the quirks TI designed into the system.

The first thing you need to know that isn't mentioned in the manuals is the bug in the DFORMAT program — it will not format the second side of a disk or in double density even when these options are selected and your hardware supports them. (Though, strangely enough, it will format SSSD 80 track disks with the new Myarc Eprom.) Thus:

*
* Prepare some formatted disks BEFORE you start working if you plan to
* use double density or double sided drives.
*

You can use any disk manager program and name the disks anything you like since the P-system does not use the first four sectors of the disk. These sectors serve only to interface with the TI system. Other versions of the P-system use this space for "bootstrap" routines to get the system started — routines that are supplied in ROM by TI. Differences between the way the P-system and the host TI system handle disks are best understood by looking at the operating system.

OPERATING SYSTEM

The P-system is not just an implementation of the Pascal language, it is a complete operating system. It has its own low-level input/output routines in 9900 machine language. The system has its own keyscan that supports ALL the ASCII control codes and screen control functions equal to many "intelligent" terminals — the system can even be set to use an 80 column terminal communicating through a serial port. Parallel and serial ports are handled just like the TI system. TI even provided an example program called MODRS232.TEXT that pokes the correct data into the necessary memory locations. The conventions for handling floppy disks, on the other hand, are unique. The system does not use the disk parameters or the bit-map in sector zero, the pointers in sector 1, or the file header space (file identifier blocks) in sectors 2 and 3, but it does write data to these sectors in a process known as "zeroing" a disk. This process fills the bit map with binary "1"'s (to prevent the TI system from overwriting the invisible P-system files), writes a single pointer in sector 1, and writes a header for a pseudofile called "PASCAL" in sector 2. Before we look at other tasks performed by the ZERO function, we need to understand a few more fundamentals such as block structure and the P-system disk directory.

Blocks

The P-system recognizes two kinds of I/O devices — character devices (such as printers, modems, and video display consoles) and block structured devices (floppy disk, hard disk, or RAM disk devices). A P-system block consists of 512 bytes -- two TI sectors. Thus disk operations read or write pairs of consecutive sectors. A disk file is a set of consecutive blocks (that's right, no fractures allowed) — in fact an even number of blocks in the case of TEXT files, though other types may be even or odd. This scheme imposes some inconveniences but also has some distinct advantages. It reduces the number of operations involved in disk I/O — no bit map checks or updates and a minimum number of track seeks in each read or write. This speeds up disk operations noticeably.

Some blocks have special functions. TEXT files are stored in "pages" of 1K (two blocks) each — that is why TEXT files are an even number of blocks in length. The actual text is preceded by a "zero page" where information used by the Editor is stored. CODE files are preceded by a single block containing data used at run time. DATA files can be used for anything else and have no special format. Bad blocks on a disk can be marked as a file with the suffix "BAD". After a disk has been used for a while, removal and rewrites of files will create unused pockets of space. These can be cleaned up with a housekeeping process called Krunching a disk. This Filer command consolidates the files by reading and rewriting them to close up unused blocks, but leaves BAD files undisturbed.

Disk Directory

The P-system uses a very compact directory structure that consumes only 4 blocks (8 sectors) on the disk. There is also an option to use an additional 4 blocks for a backup directory in case the main directory is damaged. Each file entry takes only 26 bytes for a file name, starting block, length in blocks (remember no fractures allowed), a type code, and the date of creation (coded into 2 bytes). This compares with 256 bytes per file in the TI system. This data begins in block 2 (sector 4) immediately following the 26 bytes that contain the Volume name and parameters (similar to TI sector zero data). A copy of the directory of an active volume is maintained and updated in RAM during disk operations and written back to disk when the file is closed. During some operations — removing a file for example — the system gives you the option of changing your mind before the directory is updated. As in most systems a file is deleted by erasing its directory entry rather than the file itself.

ZEROing a Disk

So now we can look at the first step in turning a formatted disk into a P-system Volume — what happens when you Zero a disk The Zero function is in the Filer program, the most-used system program after the Command processor (the part of SYSTEM.PASCAL that calls other system programs). The zero function prompts for a volume name and size in blocks. The size sets the limit on blocks that the system will access, but if it is larger than the number of physical blocks actually on the disk, the missing blocks will generate errors when you attempt to read or write to them. Whatever size you choose the entire bit map will be filled in, so unused sectors are still not available to the TI system. (More on this later.) Once these data are entered, the Zero function checks other drives for the same volume name, fills the bitmap, writes the PASCAL pseudofile header, then writes the name, size, date, and an end of directory marker into block 2 (and also into block 6 if the duplicate directory option is used). From this point on, any previous TI or P-system directory data on the disk becomes inaccessible and the system treats the remaining blocks on the disk as if they were empty.

THE SYSTEM FILES

Now that you know a few of the chores necessary to get started, lets look at what the system does for you. The P-system contains a number of special programs: SYSTEM.PASCAL, SYSTEM.FILER, SYSTEM.EDITOR, SYSTEM.COMPILER, SYSTEM.ASSMBLER, SYSTEM.LINKER, SYSTEM.LIBRARY, and SYSTEM.SYNTAX. All except the last are code files but do not carry the ".CODE" suffix required for user generated code files. Most of SYSTEM.PASCAL is supplied by TI in ROM along with data files to set system parameters, e.g. console or terminal configuration, and define the the character set. If files named SYSTEM.MISCINFO and SYSTEM.CHARAC are present on the boot disk, they are automatically read at boot up and replace the data supplied from ROM. If the short SYSTEM.PASCAL disk file is present, it is merged with the main program from ROM to provide a welcome message and textual error messages in lieu of the ROM error codes.

Boot Sequence

When your TI99 is turned on or reset with the P-code card on, the system monitor will try to boot the P-system unless it is told not to — literally — it looks for the word "NO" (ASCII codes 78,79) at integer address 14586. If these values are not found the routines in the P-code ROM are loaded and begin execution. (When you Halt the P-system, these values are replaced at that address.) The program writes system tables into RAM and then polls the disk drives both to determine which are "on-line" (i.e. have a disk in them) and to locate all the SYSTEM files. When this is complete, the welcome message is displayed and the program SYSTEM.STARTUP is executed if available. When this program (or any other) finishes, control is returned to SYSTEM.PASCAL which displays a "command line" showing the prompts for the single character system commands. Some of these commands are in SYSTEM.PASCAL (ROM), but the more elaborate ones, such as E)dit, F)iler, and C)ompile, call other programs that overwrite SYSTEM.PASCAL in memory. In fact some are so long that parts of them are paged into memory only when needed using the P-system's automatic memory management routines (the system uses "virtual" memory to overcome RAM limits -- it was DESIGNED for small systems).

Running the system

Now you are ready to write your own Pascal or Assembly Language programs. There are also some public domain or commercial programs and a few exotic ones with murky origins and no guarantees. Some users have ported versions of a Fortran compiler to the TI, but establishing your right to use such a program is tricky, since it is copyrighted and has never been released in a TI99 version. I understand that it was ported by people who purchased the original for the exclusive purpose of using it on the TI. There are also one or more versions of PILOT which was under development by TI when they pulled out of the home computer market. I've seen one of these which did not look like a finished system, but did function.

The assembler supplied with the system is a Macro-assembler several cuts above the version supplied in TI's Editor/Assembler package. My friends who work in A/L speak highly of it. The SYSTEM.LIBRARY supplied by TI contains precompiled routines to access graphics, sound, and speech capabilities of the TI99. The implementation of UCSD Pascal is nearly complete and supports program chaining and concurrent processes. Running several programs at once slows down execution and must be managed by events defined within the programs rather than interrupts, but it opens up possibilities not available in other high level languages on the TI99. These features coupled with the ability to run the system from an 80 column terminal give the TI99 a much more sophisticated feel. The ultimate limit is memory — the RAM available on the TI requires programming techniques that use lots of paging code into memory from disk, thus slowing down execution.

Public Domain Software

In spite of memory limitations, some excellent programs have been written by users. Perhaps the most sophisticated are those from Andy Cooper, particularly his terminal emulator and his GPL Disassembler. There are now several disassemblers for GPL, available or "in the works", but Andy's was available two years ago! The terminal emulator is in its second major version with some enhancements that Andy graciously added to solve some problems I had using my TI as a high speed terminal for a Pascal Microengine. Dave Ramsey and Mike Lambert of the Mid-Atlantic Ninety NinERS UG have written many useful programs including character sets, a memory reader (PEEKIN) and various utilities. Everyone has some version of the FASTBACK cloner — mine is modified to handle all formats including 80 track -- in fact it is presently the only program I know that will clone an 80 track disk. Mike King, whom I haven't met, solved the problem of importing DV80 text into the P-system TEXT format. And if you want a veritable sea of Pascal code, join USUS the UCSD Pascal user group and you can access the seven megabyte member library, most of which has never been adapted for the TI P-system.

COMMUNICATIONS

Communications was one area TI and the developers of the P-system left alone, and for very good reasons. High performance communications software -- terminal emulators for example -- usually require native machine code for critical portions to assure adequate speed. REMTALK was an early program written in Pascal to establish remote links between two computers, but it is too slow for day-to-day use and must be running on both machines (i.e. it only communicates with itself). Nevertheless, it was and is used to transfer files between different machines running the P-system.

Andy Cooper changed all this for the TI99/4A with his terminal emulator TEP. Here was an efficient communications program for the TI P-system, with machine language modules running from a Pascal host program. But the best news was its capability for binary transfers using the XMODEM (checksum) protocol. This made it possible to transfer both CODE and TEXT format files between TI99's running the P-system or to bulletin boards with XMODEM (checksum) capability. Andy wrote TEP to encourage scattered P-system owners to trade files. For those of us struggling with the system, it was the same kind of breakthrough as Paul Charlton's XMODEM program on the TI Forum. It also shared a common frustration of first-time XMODEM users -- how do you download the more sophisticated program when all you have is the TE2 package supplied by TI? Paul supplied XMODEM in a form which could be captured as an ASCII file and run from Extended Basic, but capturing a P-system CODE file (TEP) was not so easy. There are utilities in the P-system that make it possible, but they are a bit tricky for the novice.

When Andy uploaded a P-system TEXT file describing his scheme for converting the TI controller to 80 track operation, the frustration on the TI Forum was almost palpable. The description of the file aroused enormous interest but the only people who could read it were those with BOTH the P-system and TEP. After listening to complaints and confusion for several days, I cobbled up a slow, crude XB program called PASTRN to convert P-TEXT to the standard TI DV80 format. The next day Andy Cooper came back with an elegant rewrite of the XBasic that provided a 4X improvement in speed. Working independently, Andy Dessoff wrote an assembly language routine which he called PSCAN to perform the critical but slow character handling operations. At the end of that week I combined PSCAN and the XBasic host program using Todd Kaplan's XBALSAVE technique so that the XBasic and machine language could be saved in a single file that loads and executes very quickly. And that's how the TI99 community got its first Pascal text file translator.

This still didn't solve the problem of downloading TEP without TEP. I took another crack at this one while working with Phil Symerly to get Pascal downloads onto the new hard-disk for his Washington DC BBS. The scheme involved a utility I called PAS>TI (inspired by a program written for the APPLE by my friend Tom Wotecki) that read the hidden files on a P-system disk and wrote exact DF128 images that were recognized by the TI system as individual files. Converting downloads back to P-system format still involved using the RECOVER utility and the Filer's "Make" command. The process was tedious but the DF128 files could be transferred with any binary protocol including TE2. We set this up on the DC board and Bill Byrne picked it up for the Wichita TIBBS using TE2 only. Of course the first files we put on the board were TEP and its docs.

It would be months before any further help would be available for novice users, but the TI Forum and some BBS's began to build their libraries of Pascal programs. The next step depended on a new program and an old one rediscovered. The new one was a utility (SPLITP) that I wrote to split up the space on a disk between a P-system volume and normal TI files. I used it to boot either the P-system or Extended Basic from the same disk, but it also made it easy to set up valid volumes no larger than required to hold particular programs. I merged this new one with PASTRN and PAS>TI into a single XB program called PUTIL. The old program was the remarkable DCOPY which captures all the information on a disk in a single IF128 file that can be restored to a perfect clone of the original disk. In September 1986, two DCOPY files created from split format disks were placed in the TI Forum Data Library. When downloaded with any Xmodem terminal emulator and restored with DCOPY the embedded P-volumes could be run immediately. One of the programs on both disks was Andy's new 9600 baud version of TEP with full VT52 emulation and other improvements.

ADVENTURES WITH NEW HARDWARE

The easiest upgrade for the P-system is the addition of a Corcomp double density disk controller. It complicates life very little, since there are only two disk formats. You can boot the system from single density disks and transfer all your files over to the higher capacity format. There is no problem reading or writing disks without using TI's sector zero data since the hardware senses single or double density (think about it -- the sector zero byte that indicates double density can only be read AFTER the hardware switches to double density!) Once the density is determined the sectors per track is determined.

If you want to use the faster and more flexible Myarc controller, things get more complicated. The hardware senses density, but the Myarc controller uses the data in sector zero to determine whether each track contains 16 or 18 sectors. Without this data the default format is assumed (16 for the 40 track system and 18 for the 80 track). When I got my Myarc card I was satisfied to run in 16 sector format, but when I got the 80 track upgrade I had to find a way to use the available formats more easily. The answer turned out to be a very simple program I called CHECK that used the low-level UNITREAD procedure to read sector zero on all active drives. You don't have to do anything with the data -- when the controller reads sector zero, it automatically adjusts to the format data for that drive. My SYSTEM.STARTUP program now goes through this drill before it sets printer name and serial port. The Corcomp or Myarc controllers will access four disk drives, but you will find that the P-system will not recognize the fourth drive. I think the limitation is hard coded into the UNITREAD/UNITWRITE procedures for low-level disk I/O.

By the way, the P-system does not like slow printers -- the system times-out while waiting for a typical daisy-wheel printer to empty the large buffer set up by the Filer for Transfer operations. If you can't afford a printer buffer or spooler (or a faster printer), you can write or acquire a simple program to send files to the printer line-by-line.

The next piece of hardware I tried to add was a Myarc 512K Ramdisk -- and discovered a few more limitations of TI's P-system implementation. If you clone a working boot disk to the Ramdisk then set it to emulate drive #1 (Unit #4), the system will try to boot from it. Most of the boot routine executes without problem until the system polls the drives - - the system reads drive #2, then #3, then comes back and reads physical drive #1. From this point on it can no longer find the Ramdisk. Again I suspect the UNITREAD procedure works only for the three physical drives (something to do with the CRU address). I still have hopes of running the Ramdisk as a fourth block-structured device. There is room in the system table for units up to 32, but most of the slots are empty. When I get some time, I'm going to write a STARTUP program that pokes the Ramdisk volume name into Unit #10. Though it may be interesting to try it with the fourth physical drive first. It still may not work if the limitation is in UNITREAD, but its worth a try. In fact getting a Ramdisk into the system should speed it up significantly because of all the virtual memory operations.

The latest expansion was to add 80 track drives. This turned out to be a real detective story. I approached it in stages and kept a 40 track drive as Unit #4 (the boot drive), but those who switched over completely to 80 track drives had their hands full. The following message to a frustrated user gives some of the flavor of the search for answers:

"Ralph,

I think I know what your problem is. As I mentioned on the phone, the P-system at boot up senses only single or double density from the boot disk -- other disk parameters are the hardware defaults (in this case the Eprom and DIP switches). What your system is expecting is AN 80 TRACK BOOT DISK! Which you don't have yet because you can't boot the system to make one -- CATCH 22 eh? That's also why your SSSD master disks won't boot either -- the system is looking for files in the space between tracks.

But don't despair. The trick with my CHECK program will work if you put it on the first track of a 40 track DOUBLE DENSITY disk. The system can always find the first 9 sectors of a single density disk or the first 16 sectors of a double density disk. Since the first 4 sectors are reserved for TI-DOS and the next 8 are used by the P-system directory, you must use a double density disk to have any space left to put CHECK on the first track. Rename CHECK as SYSTEM.STARTUP and it will automatically execute at the end of the boot sequence. Then you can create a true 80 track boot disk in your other 80 trk drive.

Don't put any other autoexecute files (SYSTEM.PASCAL or SYSTEM.CHARAC) on your 40 track startup disk -- the system can't read them until CHECK executes. It will try to load SYSTEM.CHARAC but will in fact read bit patterns from the wrong sectors as the character definitions -- makes for an unpredictable display! There is a slight chance that your drives will not read from a 40 track disk while in default 80 track mode. If this happens, send me a message and I'll make an 80 track boot disk for you.

By the way, the reason I keep a 40 track drive in the first slot is not because of P-sys quirks but because some copy protected software crashes on 80 track drives.
Jerry"

[Ralph called a few days later to tell me it worked and marvel at the complexity of the system. I've put together a "universal" boot disk that should work with any double density drive/controller configuration -- it involves duplicating files read before CHECK is executed during the boot process so that the backup copy will be correctly read if the "wrong" sector/track value is used. The next time I open up the box I'll switch an 80 track drive into the first slot and test it. Until then, happy hacking.]

"TAKE TIME TO PRETTY UP YOUR PROGRAMS.

"Pretty Programs Bloom Forever"

YOU ARE THE PILOT. Teaching others using your computer

by Willaim Harms

Programmed Inquiry Learning or Teaching

Although I've just spent a few days learning about PILOT, I can really write a useful, enjoyable program. This language is EASY. It doesn't have many of the capabilities of TI BASIC, but it does have others not found in even TI Extended Basic.

Thomas P. Weithofer sent me the program PILOT 99, and documentation. He developed this TI99/4A version with help from Texas Instruments, Cin-Day Users Group, and Xavier University professionals. It's copyrighted 1985 by Thomas Weithofer and portions of the manual are by permission of Texas Instruments. It is a public domain package that costs one only about $10.00 plus 2 SSSD disks. What a great value! [Ed: Thomas Weithofer passsed away at the age of 20 in early 1986. His gift to the TI community will live on and can be obtained from UGN by registering this copy of the book with the bound-in registration card in the back of the book.].

PILOT was largely created by John A. Starkweather, Ph.D. at UCSF starting in 1962. In 1973 national standards were developed for the basic commands (only 8) and syntax, and now one can get a version of PILOT for most personal computers. It was developed on a small computer to be able to function completely on a small computer. Dr. Starkweather wrote a short book, which I've found to be the perfect guide. It's called, "A User's Guide to PILOT" and ublished by Prentice-Hall, Inc. at Englewood Cliffs, New Jersey 07632. I ordered it at the local B. Dalton Bookseller.

I would evaluate the TI version as one of the best teaching aids available in the world of software, since it's easy to write programs and offers most all of the features that make a lesson useful and enjoyable. The only feature I would like to see added is that of Speech.

PILOT 99 seems to be written in TI-Forth and thus a program can run pretty fast. It shows the power and versility of TI-Forth. While one is thus limited to a small program running at one time, one can run programs quickly with each drawing needed data from files the other programs have created.

To use the version of PILOT 99 that I got, you will need TI's Editor/Assembler cartridge, expanded memory, a disk system, and a word processor that can create display/variable-80 (text) files. You would write the program in the word processor just like the big computers/software use , which is nice in some ways since with one like TI-Writer you've got a full screen editor and other useful commands available. Then you would fire up the Editor/Assembler and use the Load and Run Option, entering DSKn.PILOT. When it is loaded enter the file name of the program you created with the word processor. The PILOT 99 software will run the program until it finds an error in which case you get an error message at that point. Thomas Weithofer says there is also a version one can use out of TI Extended Basic.

PILOT 99 adds many commands beyond the basic PILOT set. You have all the normal TI Extended Basic Sprite Commands, which provide great enjoyment to user and liven the presentation of any subject matter. Thomas has also ad the Joystick commands, TI's character graphics commands with color, real live Bit Map Graphics ie, Draw Circle, and Mass Storage device commands for files usage.

The manual is excellent, all 70 pages of it (on disk). Each command is described and an example given in a program context. However, it says that data files are Internal Fixed 80 Relative Update, but the file I got when writing data out to disk was Display Fixed 80. To help me use the manual I created a kind of Table of Contents and Index.

Bit Map graphics are easy to create and are displayed in the top 2/3rds of the screen with the bottom 1/3 reserved for full sized text. In the top 2/3rds graphics area you can also display text, but it will be smaller(64 characters per line). The command for Draw Rectangle is: DR: rowl, clml, row2, clm2, ie. DR:50,50,100,100 will draw a rectangle with the top left at position 50,50 and the bottom right at 100,100. Then one could use the command "T:Thats a rectangle, folks!" to produce the message at the bottom of the screen. Better yet, to describe the language, you could ask the computer operator ie. student some questions about the rectangle. Here's a really short program to illustrate.

By the way, PILOT doesn't use line numbers. It's like LOGO, LISP, and some other advanced languages in this respect. One uses labels and subProgram like techniques to structure the program and direct the flow of action.

```
R: Remark only - prog. to demo a Q & A.
IG:
DR: 50,50,100,75
TG: 1,5,shape is 50 by 25 units
T: how high is that rectangle?
A: #A
M: 50,50 UNITS
TY: That's perfectly correct
TN: Nope, thats not just right
T(#A=25): You were thinking of the WIDTH
T: press any key to proceed
```

```
R: is for a REMark
IG: is to Initialize Graphics
TG: puts the text at row,column used
T:  is to Type something to the screen.
    (TP: is to Type to Printer)
A:  is to Accept an Answer
M:  is to Match to the following possible strings
    each seperated by a comma
TY: is to Type only if the previous Match was True
TN: is to Type only if the previous Match was Not-true
T(#A=25): is to Type only if the expression is True
          (here users answer of 25 would be true)
```

Instead of the TY: and TN: we could have used a command- JM:*LABEL for Jump-on-Match to a label. After the *label would come some testing routine that ended with an E: command to return the program flow to the line following the JM:*label.

We could have used the Match or Jump command- MJ: string-to-match,more. If no match is found to the strings in the statement, the program jumps to the next M: or MJ: statement.

User subroutines are invoked with a simple- "U:*YOUALL" (U:*title). They are also ended with the command- E:. Problems can be identified with the PR: command, then you can jump to them easily. You can put the Y or the N or the conditional expression ie,(#A=25) after any of the basic commands.

To save that answer to a disk file we would just add a command- Write Answer- WA: right after the A: in the program above. Earlier in the program you would have the command to open the file- OF: DSK2.FILENAME or some other file and then later would close the file with- CF:.

For math you use the C: (Compute command) with the characters <- instead of the = sign. For example: C: #F<-88 or C: #E<-#G. The first sets F equal to 88 while the second sets E equal to the value of G. All the other TI numeric operators ie. + are available as are the numeric functions such as TAN for Tangent.

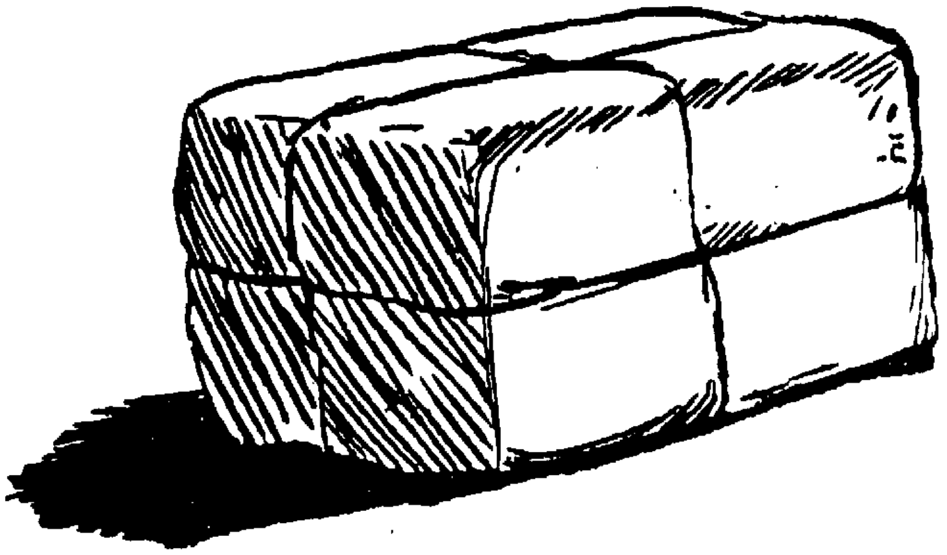
PILOT is for easy interaction between the computer and the user. A simple example of it is:

```
T: Enter your name
A: $A
T: Enter an adjective
A: $B
T: Enter a type of animal
A: $C
T: Enter a part of an animal
A: $D
T: Enter a color
A: $E
CH: (this means Clear-Home the cursor)
R: * * * *
T: $A had a $B $C,
T: whos $D, was $E as snow
T: Everywhere that $A went, the $C
T: was sure to follow.
```

There are many other commands in PILOT 99, but most are just like TI Basic or the Sprites in TI Extended Basic. Most are easy to remember and there are only 54 with the 1 or 2 digit code. I've barely scratched the surface in this memo of the many ways the commands can be combined to produce a very enjoyable interactive session of learning or data collection. Dr. Starkweather describes many in his book.

--- E X P L O R E ---
in Harms Way

INDEX OF PILOT COMMANDS FOR THE TI-994/A
PREPARED BY BILL HARMS



Exploring Your Hardware Package

LOAD INTERRUPT, HOLD and RESET SWITCHES FOR THE TI 99/4A COMPUTER

by Brian Kirby
Compuserve ID 70346,1703.

First, let's describe what each of these switches will do for you and the computer:

LOAD interrupt: The load interrupt, when activated will cause the computer to suspend its current operations. Then it will look at a specific memory locations that will tell the computer where to go for the next set of directions. This switch is useful for several utility type programs. You can have a debugger or disassembler loaded in the memory along with the program you plan to check. When your running program cuts up, you can hit the load interrupt and be put into your debugger program. Then you can go see what happened to your program in the computers memory. Another use is screen dump routines. You can have a utility loaded up in the computer and your program. When you want a copy of the screen you hit the load interrupt switch and then the screen dump routine takes over and you end up with a hard copy of what was on the screen. You can come up with all kinds of utilities for the load interrupt switch.

In specific a load interrupt causes the 9900 cpu to initiate a interrupt sequence immediately following the instruction being executed. The memory location at >FFFC is used to obtain the vector for the Workspace Pointer and the Program Counter. The old Program Counter (PC), Workspace Pointer (WP) and the Status Register (ST) are loaded into the new workspace and the interrupt mask is set to >0000. Then the program execution resumes using the new PC and WP.

Here is a check, just for grins, that will let you know that the load interrupt works. If you have a memory editor type program (SBUG,MEMORY+AID, GRAM KRACKER,etc) go into memory location >FFFC and change the next four bytes to >83 E0 00 24. The first two bytes are the Workspace Pointer (>FFFC) and the last two bytes are the Program Counter (>FFFE). If you do a load interrupt using these changes the computer will do a power up reset routine. Another is to set the WP and PC to >83C0 and >0900. This is a level one interrupt. When you do it, the system will lock up, but you will note all your P-Box cards lights will be on except the memory.

HOLD: The hold does what it implies. It puts the microprocessor on hold. It's good for stopping the computer dead in its tracks. Works great for games that do not have a pause function. There is times when you do not want to use it. The states you do not want to be in are Input/Output functions. Mainly, like during a disk read or write or initilization routine. I think you can understand why, but if you don't know its possible to crash your disk or cause some timing problems during a file transfer. Let's not worry about that. The real uses for the hold, is so that other devices may access the computer busses without the 9900 CPU on line.

Specifically, when the hold is active, it is signaling the CPU that an external device, such as another CPU or a DMA device would like to use the address and data busses to transfer data to and from memory. The 9900 goes into the hold state when it has completed its present memory cycle. The 9900 then places its address and data buss tranceivers into an high impedance state, along with the control lines WE,MEMEM, and DBIN. Then the 99C will activate another signal called HOLDA. This is a hold acknowledgment. When the hold is removed the processor will return t normal.

After installing the hold switch, it is very easy to test. Just turn it on while listing out a program in basic or XB. Try it during a game.

RESET: Again it resets the computer. It causes the computer to do the power up routine. This is great when the computer locks up. You hit the reset switch and your back to the title screen. This saves wear and tear on your power switch and extends the life of the computers power supply. There have been many articles on the reset switch and not all reset switches work properly. Let me explain why. First the basic form of the reset comes from the cartridge that you plug in the computer. There is a line that runs from the GROM port or cartridge port back to the clock chip that supplies timing for the whole computer. When the GROM port reset line goes low it causes the clock chip to reset and it in turn passes a reset on to the CPU and the 9918 VDP and the 9901 CRU chips. If you have a Widget this is what they use to reset the computer when you put a new cartridge in. But I'm sure you have notice that when you have locked up a few times and the reset on the Widget didn't do the job, You had to shut the computer off and on to bring it back up. This was due to a lockup in the clock chip and it could not pass the reset along.

First the required parts:

One push button switch,normally open type, use a micro type if you pl to mount it in the console.

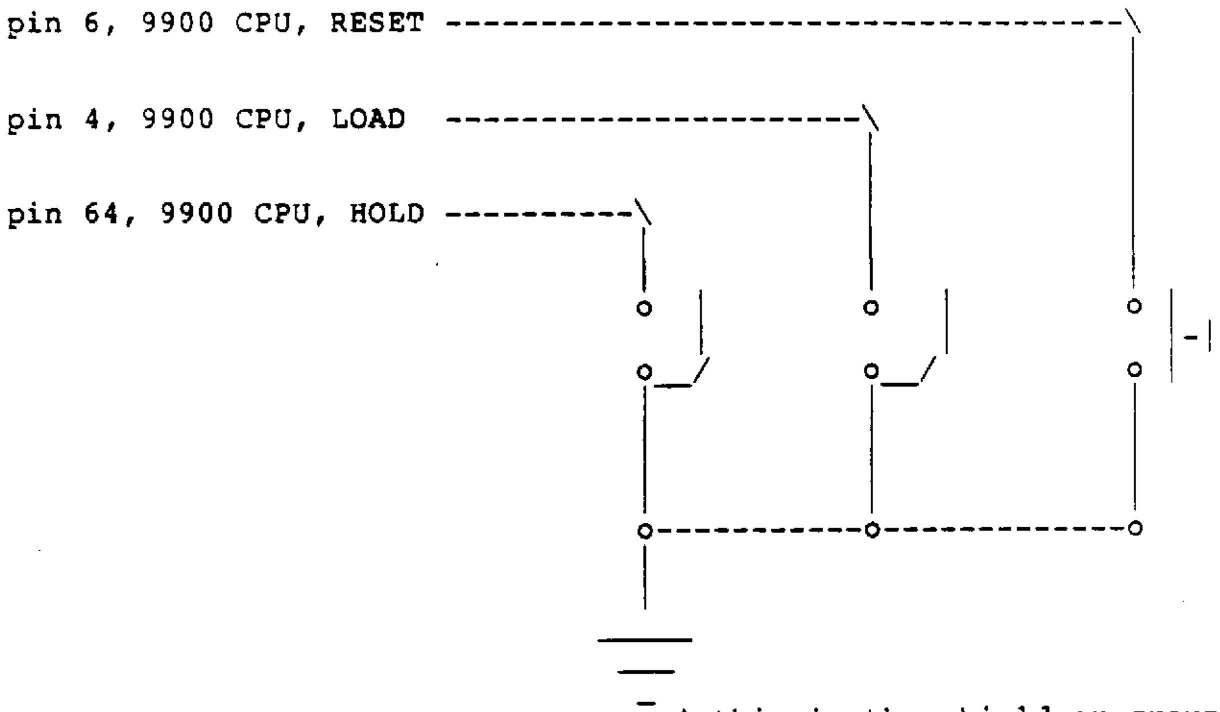
Two lever type switches, normally open, again micro types if for the console.

Three 2.2 uF/16V tatalum capacitors.

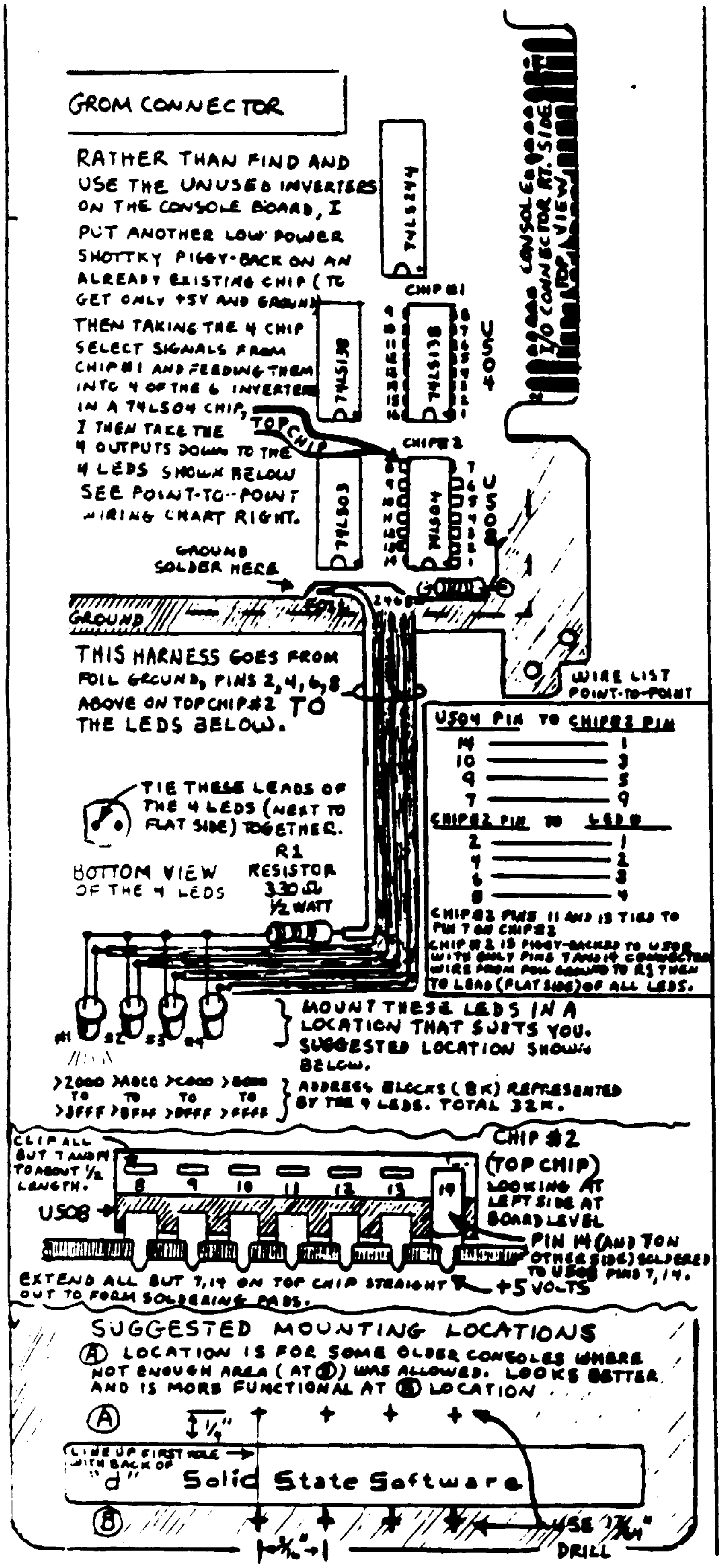
About 4 feet of small gauge wire for hook up. Wirewrap wire is great if you mount the switches inside the console. If you want to not drill holes in the
console, buy some ribbon cable and a mini box.

Open up the console ~~by remove the screws on the~~ bottom of the console. Note how the door on the I/O port to the P-Box is installed. Then note how the power switch is assembled on the power supply. Remove the screws on the power supply board and set the power supply aside. Remove the plug from the power supply to the computer board. Note how the plug connects. Notice the keyboard and how it connects to the computer. Remove the screws that hold the keyboard and remove the keyboard. Th computer is then removed by taken out the remaining screws that secu it. Note its position. Then remove the screws that hold on a shield to the I/O port. Note how that connects. Then remove the remaining screws that hold the shields on the motherboard. Locate the 9900 chip inside. Its the biggest chip and it has 64 pins. On the bottom of the board, where no ICs are mounted, locate the CPU chip. We are interested in pins 4 (LOAD), 6 (RESET) and 64 (HOLD). Solder three wires to these pins mark the wires as to what they are. Be very careful not to splash so or to short out connections while soldering. Bring these wires out thru a hole in the shield. If you are going to install switches in the console, come out thru a lower hole near the power supply. Reassemble the motherboard with its shields and note all the above that was discussed while taking it apart. If you are going to mount the switches in the console a good place is beside the power supply so the switches stick out beside the I/O port. Be sure to mount them so that they do not short to the power supply and make sure you will have enough room to mount your speech synthesizer. If you are using stand alone devices, you may want to mount the switches in the rear of the console. Now that you have found a location that works, mount the switches and solder one each of the three wires to each of the switches. Make sure that the reset line goes to the pushbutton. Solder one of the capacitors to each switch across the connections. Make sure the positive side of the capacitor is connected to the line that goes to the computer. On all of the switches run a jumper to the other side that has no connections. Jumper all of them together and run one wire back to a ground on the computer. The shield is a good ground point. Put the computer back together following the reverse of taken it apart.

If you do not want to drill holes you have several options. First you can use ribbon cable and run it out of the rear of the computer to your minibox where you can mount your switches. This way if you decide to remove the switches you can just unsolder your connections and everything will be back to normal. You can also mount the load interrupt switch external to the console, by coming off of the I/O port. You can mount the switch in the speech synthesizer be connecting one side of the switch to edge connector finger number 13 (LOAD) and the other side of the switch to pin 21 or 23 or 25 or 27 (all grounds). But you cannot access the hold or reset thru the I/O port. They do not make it outside of the computer. If you want just a load interrupt, Navarone sells a board that goes between the "firehose" and the console and supplies a load interrupt. Its about $15.



* this is the shield or ground connection



RATHER THAN FIND AND USE THE UNUSED INVERTERS ON THE CONSOLE BOARD, I PUT ANOTHER LOW POWER SHOTTKY PIGGY-BACK ON AN ALREADY EXISTING CHIP (TO GET ONLY +5V AND GROUND) THEN TAKING THE 4 CHIP SELECT SIGNALS FROM CHIP#1 AND FEEDING THEM INTO 4 OF THE 6 INVERTERS IN A 74LS04 CHIP, I THEN TAKE THE 4 OUTPUTS DOWN TO THE 4 LEDS SHOWN BELOW SEE POINT-TO-POINT WIRING CHART RIGHT.

THIS HARNESS GOES FROM FOIL GROUND, PINS 2,4,6,8 ABOVE ON TOPCHIP#2 TO THE LEDS BELOW.

TIE THESE LEADS OF THE 4 LEDS (NEXT TO FLAT SIDE) TOGETHER.

U504 PIN	TO CHIP#2 PIN
14	1
10	3
9	5
7	9

CHIP#2 PIN	TO LED #
2	1
4	2
6	3
8	4

CHIP#2 PINS 11 AND 13 TIED TO PIN 7 ON CHIP#2
CHIP#2 IS PIGGY-BACKED TO U508 WITH ONLY PINS 7 AND 14 CONNECTED WIRE FROM FOIL GROUND TO R1 THEN TO LEAD (FLAT SIDE) OF ALL LEDS.

MOUNT THESE LEDS IN A LOCATION THAT SUITS YOU. SUGGESTED LOCATION SHOWN BELOW.

>2000 TO >3FFF	>A000 TO >BFFF	>C000 TO >DFFF	>E000 TO >FFFF

ADDRESS BLOCKS (8K) REPRESENTED BY THE 4 LEDS. TOTAL 32K.

CLIP ALL BUT 7 AND 14 TO ABOUT ½ LENGTH.

PIN 14 (AND 7 ON OTHER SIDE) SOLDERED TO U508 PINS 7,14.

EXTEND ALL BUT 7,14 ON TOP CHIP STRAIGHT OUT TO FORM SOLDERING PADS.

SUGGESTED MOUNTING LOCATIONS
(A) LOCATION IS FOR SOME OLDER CONSOLES WHERE NOT ENOUGH AREA (AT (B)) WAS ALLOWED. LOOKS BETTER AND IS MORE FUNCTIONAL AT (B) LOCATION

WHISTLES AND BELLS ARE NICE BUT LIG

I've been putting memory in consoles and speech synthesizers fo nearly a year now and can account f about 70 such units out there, som them being in very distant and far places. Well, ONE person (ED MENASIAN) said he'd like to know when his memory was functioning, since with the PEB unit now ~~removed,~~ there is no flashing LED to indicate that the memory is in operation. I've come up with and re-fined a pretty "FLASHY" upgrade to the console or speech, which will display not only the fact that the memory is functioning, but exactly which 8K block you are in at that instant.

The drawings to the left of this text, will, if you take a few moments to study, explain how to install the unit inside of ANY console, except the very few QI consoles that TI produced. (These consoles are Identified by the CPU chip being mounted vertically on the main board, rather than horizontal-ly.)

I have not included, because of space, drawings for the speech, but the same proceedures apply conceptually.

PARTS LIST:

- About 10" ribbon cable (at least 5 conductor)
- 1' of single conductor, 26 or 28 guage insulate wire.
- (1) 74LS04 chip.
- (4) standard size LEDs what ever size suits yo
- (1) 330 ohm 1/4 watt resistor (or approximate).
- Phillips screw driver (#2 tip size), 15 to 25 watt grounded soldering iron, thin resin core solder, wire cutters/strippers, patience.

After you have gathered the above items, remove the console board, and taking the 74LS04 chip in hand, bend the pins, 1 thru 6, and 8 thru 13 out so they are on a flat plane 180 DEG. in reference to each other. Now snip the smaller extensions of ALL pins off.

Set the chip down on the U508 chip as shown in the drawings to the left, and solder pins 7 and 14 to the cor-responding two pins on the U508 chip. You may desire to put a drop of super glue on the top of the bottom chip. and hold the new chip (TOP CHIP) on in correct position for awhile. This es the two chips a very firm pair.

Now just wire as shown in the wire list on the left and mount your LEDs.

BY THE WAY you don't need memory in your console or speech for this modif-ication to work, it will work for any 32K even if in the PEB!!!!

YOU DO ACCEPT FULL RESPONSIBIL IF YOU DESTROY YOUR CONSOLE!!!

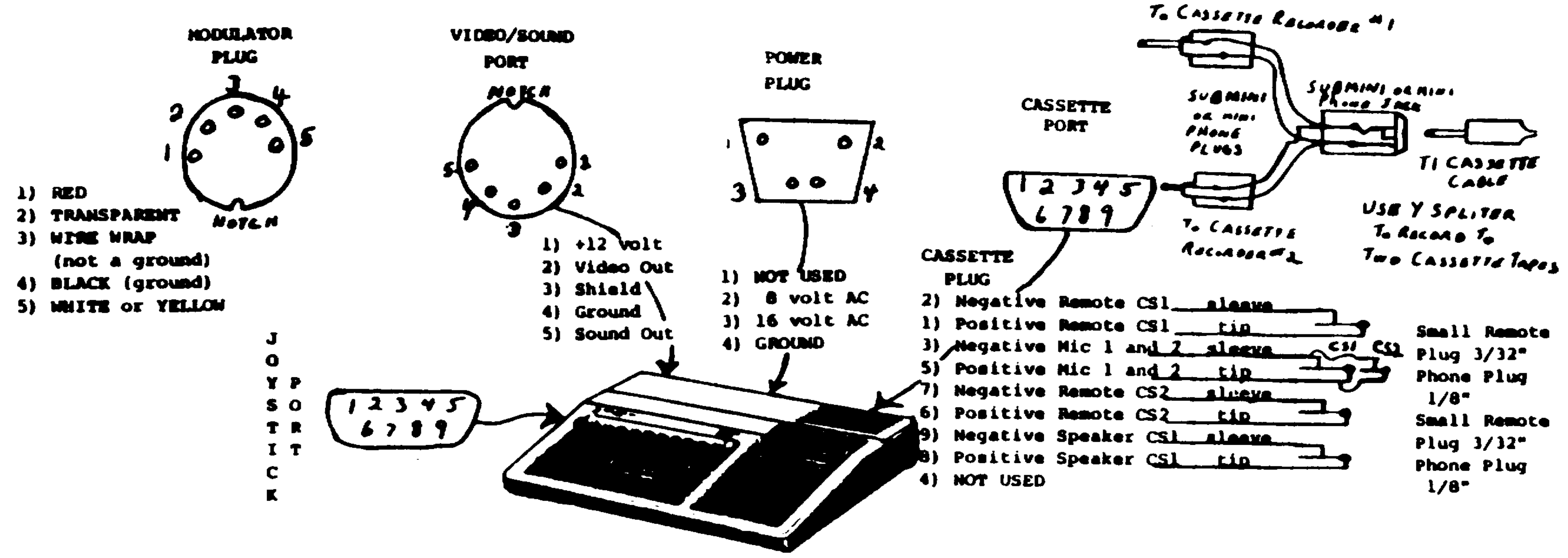
HA
FU

JOHN F WILLFORTH

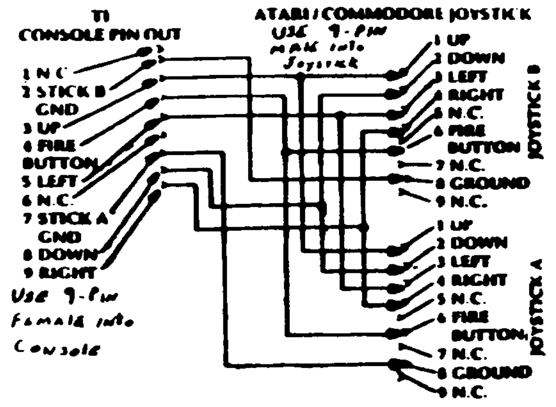
W I R I N G D I A G R A M S [illegible] P I N P O S I T I O N S

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

MARSHALL

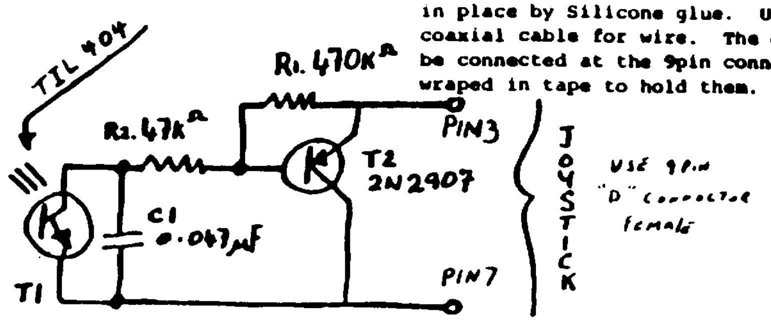


MODULATOR PLUG
1 2 3 4 5
NOTCH
1) RED
2) TRANSPARENT
3) WIRE WRAP (not a ground)
4) BLACK (ground)
5) WHITE or YELLOW
VIDEO/SOUND PORT
NOTCH
1 2 3 4 5
1) +12 volt
2) Video Out
3) Shield
4) Ground
5) Sound Out
POWER PLUG
1 2 3 4
1) NOT USED
2) 8 volt AC
3) 16 volt AC
4) GROUND
JOYSTICK PORT
1 2 3 4 5 6 7 8 9
CASSETTE PORT
1 2 3 4 5 6 7 8 9
CASSETTE PLUG
2) Negative Remote CS1 sleeve
1) Positive Remote CS1 tip
3) Negative Mic 1 and 2 sleeve
5) Positive Mic 1 and 2 tip
7) Negative Remote CS2 sleeve
6) Positive Remote CS2 tip
9) Negative Speaker CS1 sleeve
8) Positive Speaker CS1 tip
4) NOT USED
CS1 CS2
Small Remote Plug 3/32" Phone Plug 1/8"
Small Remote Plug 3/32" Phone Plug 1/8"
To Cassette Recorder #1
Submini or mini Phone Plugs
Submini or mini Phone Jack
TI Cassette Cable
To Cassette Recorder #2
Use Y Spliter To Record To Two Cassette Tapes



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 wraped in tape to hold them.



Technical Talk

HI-RESOLUTION MONITOR

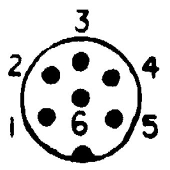
Having expanded my collection of 99/4A's to 2 and also having an old Hi-Resolution monitor ex an X-Ray Medical T.V. system. I decided to connect the black and white monitor to the "Y" signal, on the socket of J2Ø1 (J2Ø1 is the 6 pin socket that feeds the UHF or VHF modulator). The "Y" signal contains all the necessary sync. & luminance levels to run a monochrome monitor.

Modification completed, I was confronted with a good picture that would not stay synchronised . Rolling vertically or horizontally with the slightest change in picture content

On examination of my UHF modulator I noted that the "Y" connection does not correspond with the circuit diagram. Reconnecting the plug J2Ø1 as per the following table produced a good stable picture.

J201

cct. diagram	* correct connection
1 = +12v	1 = +12v
2 = R-Y	2 = Y
3 = Audio	3 = R-Y
4 = Y	4 = B-Y
5 = B-Y	5 = Audio
6 = Ground	6 = Ground



Rear view of plug.

Plugs are easily obtainable from Atkins Carlyle. Plug type is DP6, cost $1.86.

* This is true for the PAL 99/4A's but may not be valid for the NTSC version.

Remember that you will need [illegible] Audio connection also, if [illegible] going to have the dulcet tones, to remind you that you have just " BAD VALUE "'ed again.

Steve Wilkinson

CALL A COLOR A COLOUR

or

The Hi-Resolution Sequel

The sequel to the Hi-Res. Monitor article came some 3 - 4 months after the conversion.
Whilst busily working on a program, I came across one of those " Software " faults that T.I. missed & everyone else has missed.

I was using the CALL COLOR subprogram. It would just not work. The picture stayed Cyan regardless.
You guessed it! Monochrome Monitor.
Well my face changed colour even if the picture didn't !

Steve Wilkinson

AUTO FIRE PROJECT

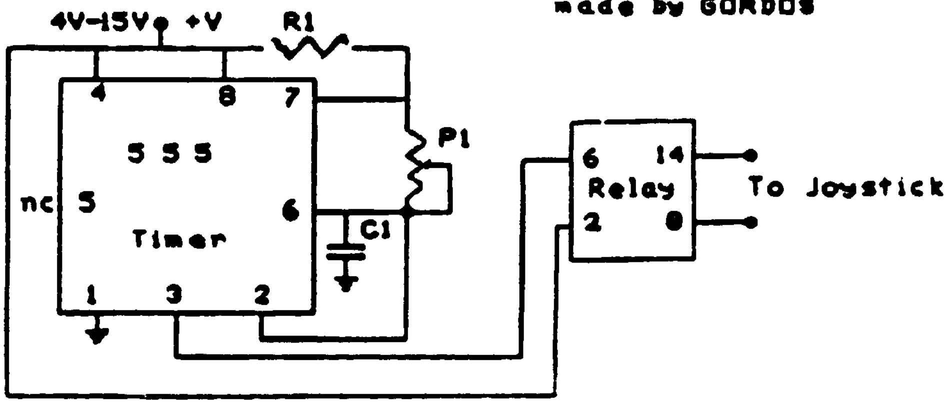
from Channel 99 Hamilton UG
by David Storey

I have been asked by several people why is it that the auto fire add on for the Atari does not work on the TI 99/4A. Well the 99 does not have any voltage output at the joystick port. It also has to have a physical contact making and breaking for the fire button to work.

This prompted me to come up with this simple circuit. It uses a 555 timer and a relay. R1 and P1 deal with the time constant. This circuit works well although it is a bare bones circuit and could be modified to give more range of firing speed but, I will leave that up to you. Here is the circuit, you will need a battery. I used a 9 volt as it is compact. This circuit as is will run with voltages from 4 volts to 15 volts.

Auto fire drawing.
by D.STOREY.

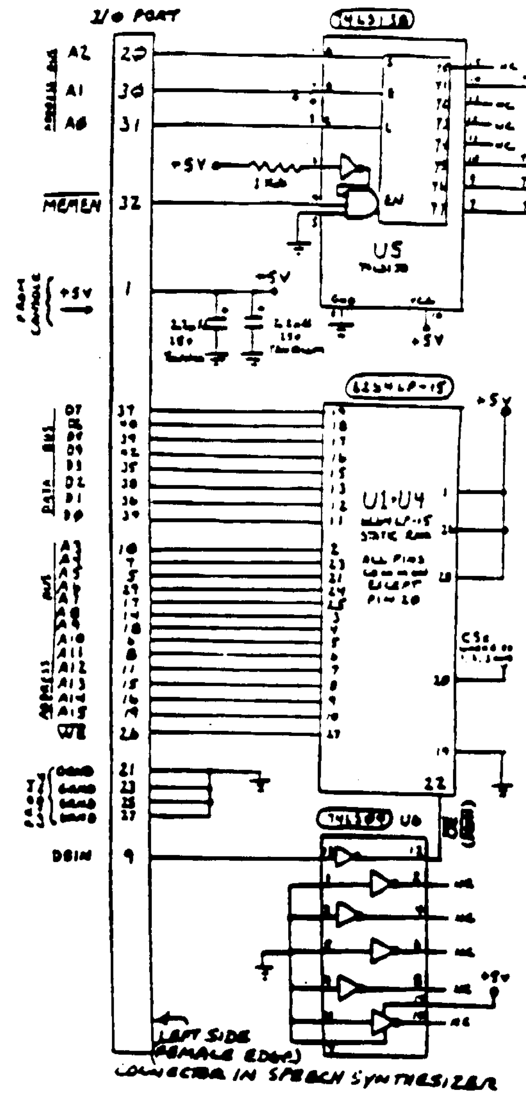
R1=1Meg
P1=1Meg lin
C1=1mfd
Relay=831C1 Dip
made by GORDOS



Opening Up Your Hardware

2K MEMORY EXPANSION PROJECT....

Here's an article that tells how to go about placing 32K memory expansion inside your speech synthesizer, stand alone disk controller, etc. I think placing the 32K in the speech box is a better place than the console.



32 KiloByte MEMORY EXPANSION FOR INSIDE THE SPEECH SYNTHESIZER (OR ANY PLACE YOU WANT TO PUT IT).

by JOHN WILLFORTH (based on ideas from the WESTRAILIA, and the CEDAR VALLEY USERS GROUPS)

I have written up several articles on the subject of putting 32K of static RAM inside of the TI console. I believe that most of the information for this came from the WESTERN AUSTRALIA U.G., and the work leading to the insertion of the same memory into the Speech Synthesizer, was done by the CEDAR VALLEY U.G.

Now I have put memory into both the console and the Speech Synthesizer. I thought that there should be no place you couldn't stick it. So I just finished putting it into the OLDE TI STAND ALONE DISK CONTROLLER (part of the old train). This made a nice quiet, sort of micro-expansion system (without RS232/PIO). If you already have a full blown system, or are just beginning to get int a disk system, and realize that you either don't have the funds, or will not need anymore than that just described, you should read on.

The long connector on the left of the schematic, represents the large 44-pin conn. that is inside the speech synth., or any other plug in peripheral ie: Stand-alone Disk Cont.. The big difference, however, is that ONLY the speech synthesizer carries pins 1,2,43, and 44 into the unit from the console. Therefore if you do decide to put memory into any other unit than the speech synthesizer, I would recommend that you wire across that unit, in other words you should run a wire from pin 1 on the console connector to pin 1 on the output end of that unit, where the 2nd unit from the console might be plugged in, and do the same for pins 2, 43, and 44. This will enable you to put the very small speech synthesizer out on the end, instead of between the 2 much larger units (console and Disk Controller). There is only one lead that is involved here that is a must, and that is the pin 1, since I have stayed with using the +5 VDC from the console, rather than tapping it from the +5 Volt source in the unit where this is installed.

If you have the documentation on the RAM chip, you may be confused by the reverse order of the address lines. DON'T WORRY, just wire the chip up as I have indicated, and if you do your part correctly, it will work. I've done nearly 20 of these installations in the console and the speech synthesizer, and in a stand alone disk controller, and as far as I know, they are all working. If you want the more simple instructions, on how to install this same memory into your console, (which is what I prefer) just contact me, by sending a stamped , self-addressed envelop, and I will send the in [illegible]uctions. Have fun! JOHN WILLFORTH RD#1 BOX 73A JEANNETTE, PA 15644 , or call after 9:00 PM, (412) 527-6656

HARDWARE HINTS

HALF HEIGHT DRIVE INSTALLATION
By Ken Gladyszewski

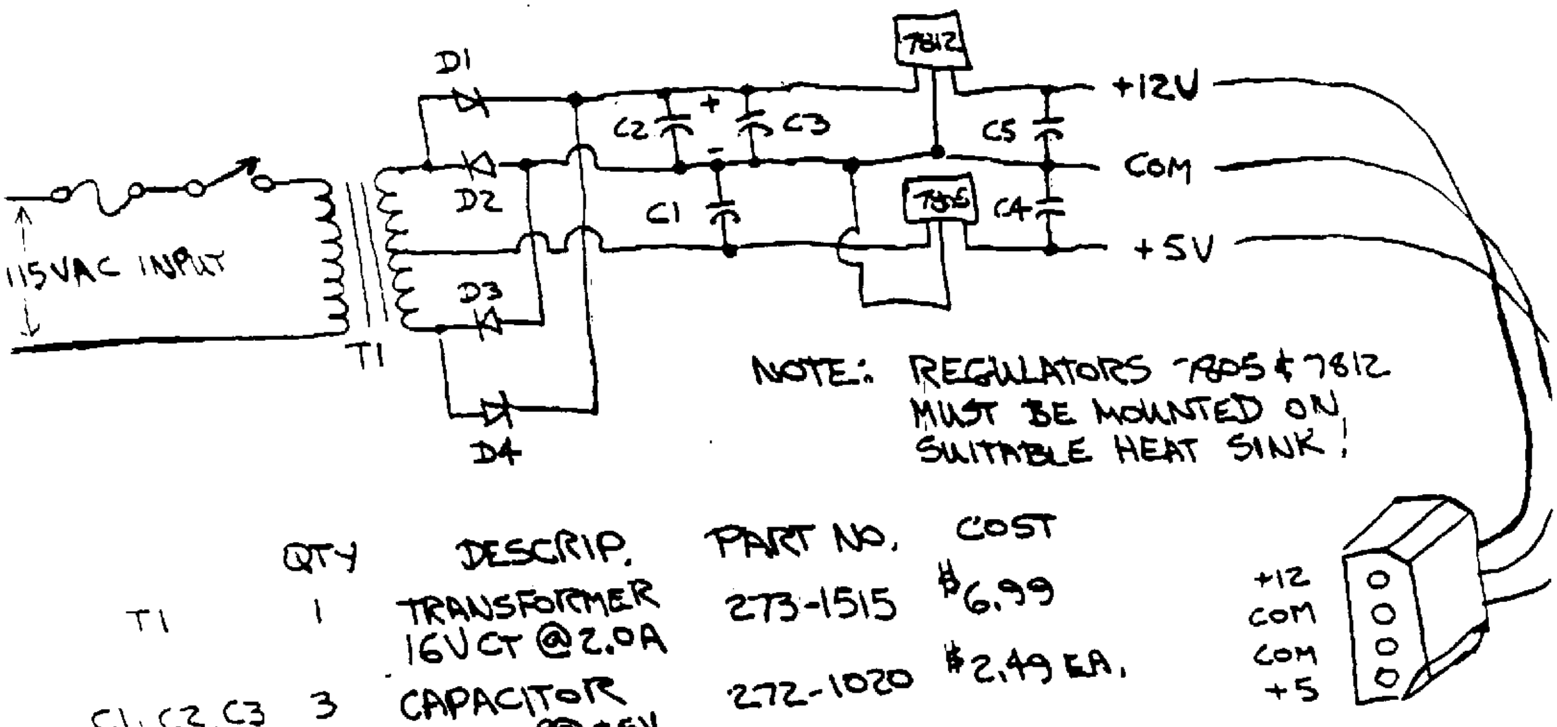
When installing a pair of half height floppy disk drives in the peripheral box, extra connectors for both the interface and power cables are required for the second drive. The interface cable can be handled by adding a 34-pin card edge connector (Radio Shack 276-1564) to the existing ribbon cable (the cable is just barely long enough), or by removing all the connectors from the existing cable and re-installing on longer cable. (Orientation to colored stripe is important. Improper installation causes drive to improperly run continuously, but with no apparent damage.)

KEYBOARD REPLACEMENT TIP
By Ken Gladyszewski

When a beige keyboard is installed in a black and silver console, great care must be used to center the keyboard to eliminate binding of the outer keys. Ron Minadeo has discovered that the overlay strip above the number keys held in a plastic extrusion which is fastened to the com with double sticky tape. It can be repositioned higher ... more key clearance by filing or sanding the upper corners of this extrusion and refastening it.

EXTERNAL DISK DRIVE POWER SUPPLY
BY Ken Gladyszewski

When I bought a case and power supply for an external disk drive, I was amazed at how simple and uncomplicated the power supply was. I present the circuit and parts list here for those wanting to build their own, because they already have most of the parts. These parts are expensive and total $21 (without a recommended on-off switch and fuse). Better and less expensive complete power supplies or these same parts can be purchased from a surplus house by mail. Sheet metal enclosures can be obtained similarly. Any power supply with 12 volts DC @.5 AMPS min, and 5 volts DC @1.0 AMPS min. should power most any single full height drive.



NOTE: REGULATORS 7805 & 7812 MUST BE MOUNTED ON SUITABLE HEAT SINK!

	QTY	DESCRIP.	PART NO.	COST
T1	1	TRANSFORMER 16V CT @ 2.0A	273-1515	$6.99
C1, C2, C3	3	CAPACITOR 2200 µF @ 35V	272-1020	$2.49 EA.
C4, C5	2	CAPACITOR 100 µF @ 35V	272-1016	.79 EA.
D1 - 4	4	3 AMP BARREL DIODE IN5402	276-1143	2/89¢
	1	5V FIXED REGULATOR 7805	276-1770	$1.59
	1	12V FIXED REGULATOR 7812	276-1771	$1.59

R.GLADYSZEWS
12-31-

"COOL IT: Help your steaming TI 99/4A
run all day long like one
reliable cool cat.

While I have not had a problem with my computer crashing because of heat, some TI owners have found that their units go beyond the point of annoyance into a twilight zone of gnashing teeth. Since I work my computer daily for several hours at a time doing word processing, I was quite concerned about the significant heat generated inside the TI/994A. I thought of add-on fan cooling, boring a hole in the bottom of my computer table and hanging a fan underneath, or moving the study furniture around to locate the computer in front of the air conditioner. These solutions were either inconvenient or uncomfortable. Besides, what about the air conditioner in the winter time? That idea was obviously not very practical. It was time to investigate realistic solutions.

Having cautiously grounded myself and a Phillips screwdriver to the kitchen water faucet (computers get a charge out of static electricity), I removed the bottom cover of my hot plate computer for a look-see. That 4 1/2" square board was the heating element for sure. I removed the two mounting screws to peek at the upper side. Good grief. There was a big black heat sink obviously positioned to reduce air flow to zero, reaching right up to the plastic top cover. That board had to come out of there. I've assembled electronic kits before. There must be an easy fix. I would even settle for a not too difficult fix.

Brousing through the local Radio Shack store, I noticed a metal chassis box with ventilating slots. A quick check of the catalog showed Part #270-253 to be 5 1/4" x 5 7/8", perfect for holding the TI's printed circuit board power supply. A few more minutes of thought, and the shopping list below was purchased and carted home. My power supply now cools itself in it's own comfortable sheet metal box with a genuine toggle switch and large amber pilot lamp. This unit has been on continuously for eight hours, becoming only comfortably lukewarm to the touch. Five more units were contructed and are now in use without problems in our local users' group.

A recent TI acquaintance reported adding more heat sink material inside his factory-stock computer, only to have the plastic door in his plug-in cartridge port overheat and soften. That report spurred me to share my most satisfactory solution with loyal TI users everywhere who certainly did not purchase their computers with the intention of perculating coffee!

Here is the Radio Shack parts list:

DESCRIPTION	PART #	PRICE
Ventilated Metal Chassis	270-253	$4.99
PC Board Standoffs	270-1391	.69
5-pin DIN Plug	274-006	1.49
5-pin DIN Chassis Socket	274-005	.59
DPDT Toggle Switch	275-607	1.79
24 gauge stranded wire, two-conductor zip cord	278-1301	50'/2.79
Small nylon cable ties	278-1632	30/1.59
Small panhead sheet metal screws and 4-40 machine screws, washers, and nuts.		

Tools required: Assorted small drills, files, needle nose pliers, pencil soldering iron, jack knife, screwdrivers, tin snips, 6 inch ruler.

1. Drill four small holes 1/2" in from the corners of the aluminum chassis and reposition rubber feet as far into the corners of the aluminum chassis bottom as possible. This will provide room for the nylon PC board standoffs to match existing holes in the power supply board.

2. Drill holes to mount the new power On-Off switch and pilot lamp on one end flap of aluminum chassis bottom. Leave top 1/2" of chassis end flap clear of fittings. Slightly relieve center hole for the toggle switch top and bottow so the switch will operate without binding. A small round chainsaw file does the job quickly and neatly. Do not mount components until after the power supply board had been installed.

3. Remove bottom of TI 99/4A. Observe, tag, and record order of four wires going from power supply to main computer board. Power supply end of these four wires may or may not have a nylon plug fitting. Unsolder wires from main computer board only at this time. Remove two screws and power supply from computer. Gently bend original red pilot lamp in toward center of board. Relieve three corner holes in the power supply board to accept the nylon standoffs, then drill bottom of chassis to match. A paper template is helpful. A slip fit is desired here, not a force fit. Do not install the PC board until all sheet metal work is completed. Take the time to remove all burrs and sharp edges. Attention to such details not only protects your fingers from possible wounds, but it will protect the electrical integrity of the finished power supply. Besides, it is that little detail done right that spells quality and pride of workmanship. Your TI 99/4A already has those qualities. Your project can too!

4. Cut two lengths (4' suggested length) of 24 gauge stranded zip cord. Strip and tin 1/16" on all ends. Tin 4 pins of your choice on 5-pin DIN Plug; solder above four conductors to 4 pins. Identify the pins and tag the wires before installing the plug cover.

5. Use small nylon cable ties ever 6" or so to bind the two zip cords together into a neat 4-conductor cable. Make a loop in the computer end of cable for strain relief, just large enough to slip over the plastic post where power supply was mounted. Bind the loop with another nylon cable tie. Allow enough length remaining to reach the 4 unsoldered holes in the main computer board with some <u>slack</u>. Solder the four wires into the main board in their <u>correct</u> order. Careful examination of the power supply board and main board will reveal one of the four holes to be a ground. That may help keep your connections correct. The other wires are +5VDC, -5VDC, and -12VDC. NO WIRES ARE INTERCHANGABLE. Keep boards, plug, socket, and wiring positively identified at all times. Place strain relief loop on plastic post and secure with original screw and small washer.

6. Drill clearance holes and mounting screw holes in opposite aluminum chassis end to receive DIN socket and the transformer power socket, which must be removed from the power supply board. Mount high on chassis end but keep top 1/2" of the sheet metal clear. I unsoldered the red, black, and white transformer wires from the board, inserted the wires through the chassis end from the outside (pointed down), and clamped the ends of the socket to the chassis with 4-40 screws washers on the outside, using lock washers 4-40 nuts on the inside.

7. Cut three 3" insulated stranded 24 gauge jumper wires; tin ends. Solder one end of each jumper into the red, black, and white position holes in the power supply board, from the top side. Install standoffs in chassis bottom and gently press board onto standoffs.

8. Install new toggle switch and pilot lamp. Solder 'red' jumper wire to one bottom terminal of switch. Solder 'black' jumper wire and one pilot lamp lead to the other bottom switch terminal. Solder red and black leads from transformer socket to corresponding MIDDLE terminals of power switch. Solder white socket lead, 'white' jumper, and 2nd pilot lamp lead together and cap with small wire nut. Route all wiring clear of black heat sink. The new power switch will be ON in the UP position. The pilot light operates on an 8.5VAC line, giving a moderately soft glow which does not glare at you.

9. Take the four DC supply voltage wires, and solder to correct pin numbers on 5-pin DIN socket, remembering that you are now looking as a mirror image of the DIN plug coming from the computer. I fed the wires through the chassis hole and soldered them to the socket before mounting the socket, because I had a 4-conductor nylon plug on the power supply end. If your board has soldered wires, you may prefer to mount the DIN plug first and then solder the wires to the plug.

DOUBLE CHECK FOUR WIRE connections throughout your cable!! Out of 6 power supply conversions, as careful as I tried to be, I messed up twice. Both computers worked correctly after the mistakes had been corrected. But it is still heart stopping, when it doesn't work right the first time. Check your ground to ground wire first, then verify the order of the other wires before turning anything on. Glue the original plastic On-Off slide switch in place with rubber cement for appearance sake (later beige TI 99/4A only). Install computer bottom cover.

10. RECOMMENDED MODIFICATION OF SHEET METAL CHASSIS TOP. The black top is steel. But the silver chassis bottom is VERY SOFT aluminum. It is easy to work, but bends much too easily when pulling and pushing plugs into their sockets. Cut two pieces of light sheet metal, 1" x 4". Bend lengthways in a vise to a right angle, 1/2" x 1/2" x 4". Drill two 1/8" holes on 2" centers in one side of each angle, and matching holes in the top of the steel chassis cover, 1/2" in from each end. Pop rivet a reinforcing angle to the inside of the cover at each end. Orient the ventilation slots in the cover adjacent to the heat sink end of the power supply board and drill through aluminum ends into the reinforcing angles for small sheet metal screws. Be sure the original On-Off slide switch on the power supply board is in the ON position. (I forgot that detail the first time around). Assemble chassis bottom to cover and secure all sheet metal screws.

Six of these power conversions have been in constant use locally since May 1994. Other members in our users' group join me in recommending this conversion project. You'll like it. And your TI 99/4A will love you for it!"

Anchor Automation Signalman Series Modems
Interfacing them with the TI 99/4A

by Scott Darling
GEnie ID TIKSOFT

After reading some messages asking for help using Anchor Signalman modems. I decided to sit down and write a short tutorial covering them all. As this is conjecture on my part, please don't hold me to all I am about to expand upon, as I had to figure out most of this myself. As some of you know I operate a BBS, CALTEX #8 in Spokane,Wa. The 'provider' of the hardware had a Mark VII modem that he had bought as a 'auto-answer' modem only to find out that one had to write the software to activate the modem. That left both of out! So I talked to Anchor about the situation And was told for $30 more could get a Mark X. So off went the modem and the check! This was in June of '84 and we had never seen or heard of the Mark X. But was told it was 'HAYES' compatible. So this is were the REAL fun began and I found out all the ins and outs! Well onto the nitty-gritty.

Mark III: This modem needs no explanation as it is a direct connect ready to go. Just plug it in and fly!

Mark VII: This modem is a auto-answer, auto-dialer 300 baud modem. The only catch is it is your software driven. Which means you, the user, write the program to make it operate! This is a major drawback, at least it was for me! But when I found out I had to write the software.....Gulp......forget it! But if you find yourself with one of these. It is not a problem , it still can be used just like the Mark III.

>>>>> EXCEPT<<<<< Now here is the fun part! A wiring change is necessary for the hookup to operate the modem.

	RS232	MODEM
PIN	1	1
	2	3
	3	2
	6	6
	7	7
	20	20

The above wiring changes apply ONLY to the Mark VII.

Mark X: The Mark X is also an Auto-Answer and Auto-Dialer EXCEPT for a small detail.....This one has the ROM software to do what you want. Whew! I finally found what I was looking for! At least I thought so. So I took it out of the box and plugged it in and of course nothing! Because I knew I needed to make the wiring changes. So I decided to make up a cable interface so as not to destroy the integrity of the modem cable. So after about $10 of solder plugs and hoods I was ready to go. Here is the cable makeup that I used:

	RS232	MODEM
PIN	1	1
	2	3
	3	2
	5	20
	7	7
	20	5
		8 is DCD

Because I use pin 8 of the modem for a Copyrighted BBS, I will Leave out that connection. Besides it doesn't go to the RS232 port!

This configuration will and does work everyday. So now I had my cable made up and hooked in and was ready to go. Right? Tried to call, BBS got a carrier, and played around in it. Logged off and tried calling back. Hmmmm no carrier tone but the modem connected me anyway.

Well to make a very long and frustating experience short, I found out the COMMANDS I was sending were not being executed. It was starting to get on my nerves at this point. So here is what I found to make the modem software perform the commands. The following programs are examples:

These are setup for auto-answer for a BBS! I use these formats on mine. When you turn on the modem, these are the defaults that will power ur

"ATCOF1H0Q0V1S0=1S1=0S2=43S3=13S4=10S5=8;cr"

Well as you can see the defaults take care of a lot of different functions for you at the onset. But you'll hardly ever use most of these. The following are the ones that will be used by most people using the modem for a BBS or auto-dialer function.

This format is what I use to set up my BBS to auto-answer. Of course I don't use these exact setting's. I'm using these for an example.

```
100 OPEN #1:"RS232" :: PRINT #1:"ATQ1S0=2S2=30;cr"
```

This tells the modem not to send result codes to the DTE (caller end<'Q1'). To answer on the second ring ('S0=2'). And the escape code is CHR$(30). This format is used by my BBS while waiting for a carrier. Also notice that the command line has no spaces, Mark X ignores the spaces, so just leave them out. The Hayes REQUIRES those spaces between commands.

So now your online and running. The next step is when the caller is done and hangs up. This next line will open a different file. With the LF off (which I normally use in the BBS anyway)

```
100 OPEN #2:"RS232.LF" :: FOR A=1 TO 1000 :: NEXT A :: FOR A=1 TO 3
    PRINT #2:CHR$(30) :: NEXT A :: FOR A=1 TO 1000 :: NEXT A
110 PRINT #1:"ATHS0=0;cr" :: RUN
```

This gives a one second delay before and after the escape command. T way I set the escape command evidently gives just enough pause between characters. Line 110 is the hangup command to the modem and also tells it NOT to answer till told to do so. The reason I use RUN is I have turned off pre-scan and my 88 sector program will recycle in 10 seconds versus the original 30 before.

There is one command that will act like turning the modem off, then on again.

```
100 OPEN #1:"RS232" :: PRINT #1:"ATZ;cr" :: CLOSE #1
```

This is the RESET command. It sets the status of the modem to ALL the defaults.

The following is a BASIC program that will write a logon file. I haven't figured out how to write a basic program that will logon and CONNECT. I think there is a way of doing it. But this one works.

```
100 OPEN #1:"DSK1.LOGON",DISPLAY,VARIABLE 80
110 PRINT #1:"1ATD"
120 PRINT #1:"1T"   OR P FOR PULSE
130 PRINT #1:"13260515"   ( 1 + AREA CODE + NUMBER  for long distance)
140 PRINT #1:"1 "
150 CLOSE #1
```

This will be saved as a file that TEII will load and run from option 2 or 3. I tried calling CIS using the output of this and it worked.

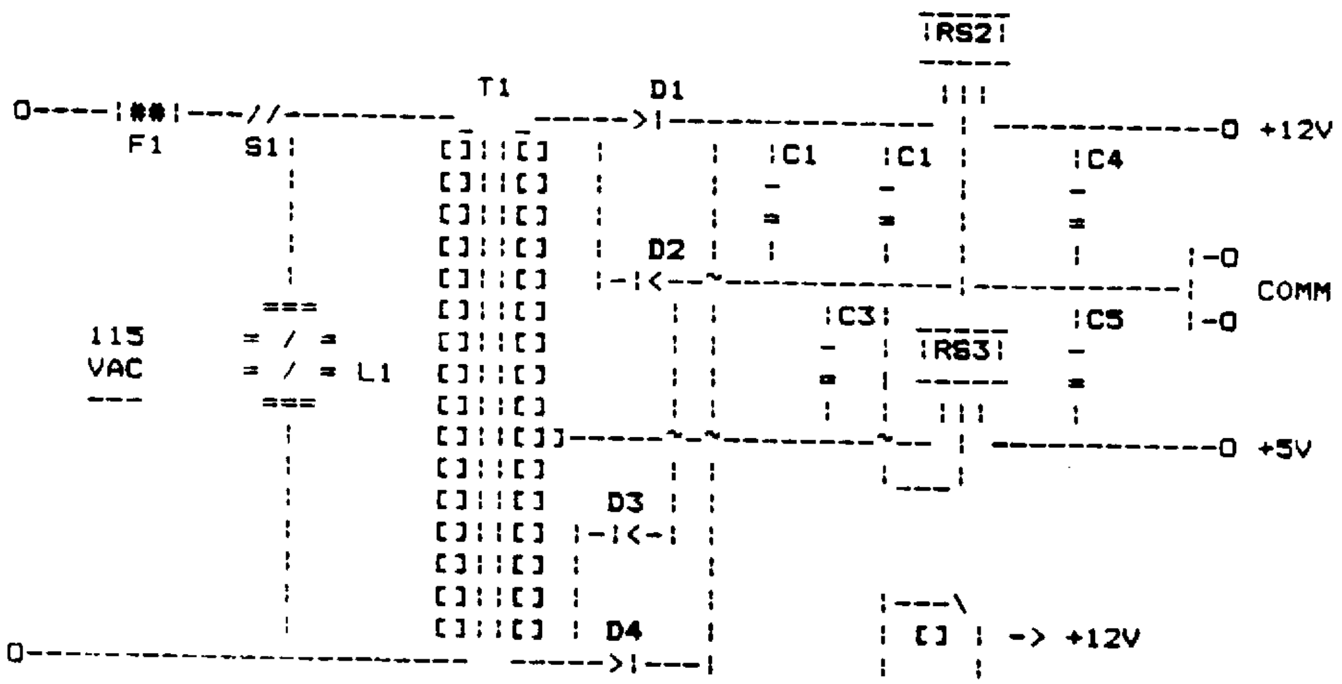
MARK XII: As far as I know all of the information for the Mark XII is the same as the Mark X. I acquired most of this info from a dealer that sells the Mark X. But as he put it "well all of the people that buy the Mark XII's wouldn't call a 300 Baud BBS anyway" so there was very little animosity between us! But from I can gather everything should be the same. Except one added command for 1200 baud

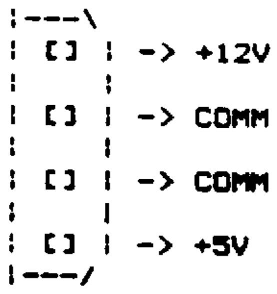
To sum up the Anchor commands: the most important part to do is the ";cr" as the modem will ignore anything sent till it receives that command. If I have caused more confusion than help let me know, as I have tried to think out this tutorial. But I could have made a mistake. This is my first try at writing(rather obvious I suppose). So here's hoping This clears up any confusion. Also, I wouldn't advise calling or writing Anchor, as they are really not equipped to handle the BASIC language to activate the commands. I know from experience.

DISK DRIVE POWER SUPPLY

This is a break down on how to construct a power supply for a disk drive. The list of parts are listed at the bottom, but remember that this list does not include the sales tax or the board to construct the power supply. If you need help in constructing this power supply just give Skip a call at 944-2770 and he will help you out as much as possible.

DISK DRIVE POWER SUPPLY





DISK DRIVE HOOKUP

NAME	TYPE	TECHINCAL NAME	PART NO.	PRICE	QT
D1-D4	3A IN5402	"BARREL" DIODE	276-1143	$ .89	(4)
C1-C3	2200UF 35V	ELECTROLYTIC CAPACITOR	272-1020	$2.49	(3)
C4-C5	100UF 35V	ELECTROLYTIC CAPACITOR	272-1016	$ .79	(2)
T1	18.0CT 2.0A	POWER TRANSFORMER	273-1515	$6.99	(1)
F1	120 VAC	CIRCUIT BREAKER	270-1310	$1.49	(1)
S1	SPST 120 VAC	ROCKER SWITCH	275-690	$1.89	(1)
L1	NE-2H120 VAC	NEON LIGHT	272-1102	$ .69	(1)
RS2	+12 VDC 7812	VOLTAGE REGULATOR IC	276-1771	$1.59	(1)
RS3	+15 VDC 7815	VOLTAGE REGULATOR IC	276-1772	$1.59	(1)

TOTAL PRICE FOR PARTS ---> $25.99

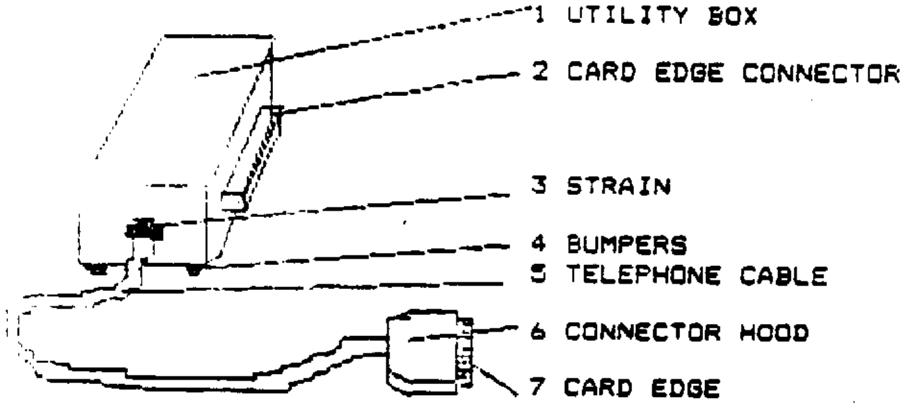
CABLE BOX

by Jim Edwards

One feature of the T.I.99 that has never been hard for me to criticize was the physical size and design of the peripheral cable and connector. It always seemed to take up an undeserved portion of desk space. With only a goal in mind and virtually no "hardware saave", I set out to alleviate the problem. It seemed a simple task to build a compact connector that would plug in without disturbing the original components. Actually, the most difficult aspect of the project was rounding up the parts.

That proved to be an education. Card edges and their matching connectors have several configurations. For example 22/44 means that it has 22 conductors on both sides. Spacings vary as well: .10, .125, .156, etc. This refers to the distance between the centers of the conductors. This project requires 44 conductors (22 on a side) with .10 centers. Finding a card edge connector was difficult enough, but finding the male counterpart was impossible. A section was literally cut out of an abandoned board.

I found most of the parts at Pacific Radio while the card was found in a card board box at All Electronics. Obviously, the exact parts may vary but be certain of the number of conductors and spacing. Once everything is rounded up, simply solder the wires together making sure to match one end to the other. Optionally, an interupt switch can be added for those screen dump programs that require one.



#	PART	MANUFACTURER	PT.#	COST
1	UTILITY BOX	CALRAD	90-785	$2.10
2	CARD EDGE CONNECTOR	GC ELECTRONICS	41-875	$4.74
3	STRAIN			.25
4	1/4" BUMPERS	RUSSELL IND.	REC-207SH	$1.79
5	50 CONDUCTOR TELEPHONE CABLE			
6	CONNECTOR HOOD	GC ELECTRONICS	41-1003	$2.48
7	CARD EDGE SCAVANGED FROM PC BOARD			$1.50

				$12.86

INSTALLATION OF GROM CHIPS INSIDE THE TI CONSOLE

by Patrick Ugorcak
OH-MI-TI

The cartridge grom chips for most of the TI modules can be installed inside the console so that it is no longer necessary to plug the cartridges into the grom port. The programs can be selected by way of a switch attached to the grom chip. This not only saves time in not having to search for a particular cartridge but it also saves wear and tear on the grom port.

Like all articles of this type I must first warn everyone that any modification to your console will void any warranty and also the risk you take is your own. If you plan on doing this modification on your only console I strongly recommend against it. There is always a chance, although slim, that a disaster might occur.

The parts you will need for this project are:

1) Program grom chips either purchased from TI for around $4 each or taken from a cartridge.
2) Ribbon cable (6 inches long, 15 wires).
3) Thin wire to connect the switch.
4) Switch (The type of switch used depends on the application. More on this later.)
5) Low wattage solder iron (25 watt or less), solder, solder bulb to remove grom chips from module if used, etc.

This project requires the removal of the grom extender, the part the cartridges plug into, from the console and attach 16 wires to it. The other end of the 16 wires are attached to the grom chips which are being installed. A switch is attached between one of the wires so that the program can be turned on and off.

What limits the number of programs which can be installed is the type of switch that is used. I have installed two programs into a console (E/A and DMII) using a SPDT type switch and see no reason why more cannot be used. One criterion for the switch is that it must have an off position so that the program attached to the grom port can be turned off when cartridges are used (extended basic for example). If you are installing only one program then any SPST switch will work as long as it is small enough to mount in the console. If more than one program is being added then a switch with an off position is needed. I used a SPDT on-off-on type switch for my two program installation. I have seen miniature rotary switches at ham meets with as many as 12 positions. Imagine 11 programs available at the flick of a switch. A mini DIP switch could also be used but may not be as convenient to operate.

Procedure

Disassembling the Console

1) Remove the on/off switch piece on the black and silver consoles.
2) Remove the 7 screws from the bottom of the console.
3) Lift the bottom part of the console from the top portion.
4) Remove the 2 screws holding the power supply to the console and remove the power supply.
5) Disconnect the power cable from the power supply.
6) Remove the 3 screws holding the motherboard to the console and lift the motherboard up slightly so that the keyboard connector can be removed.
7) Disconnect the keyboard and lift the motherboard out.
8) Remove the grom extender from the motherboard.

Preparing the Grom Chips

The grom chips will be piggy-backed together to form a grom stack. Pin 14 on each program grom chip group is attached to the switch position so that the different programs can be selected. Some of the programs use as many as 5 grom chips. For example Editor/Assembler uses 1, Multiplan uses 5 and Disk Manager II uses 2. In the case where more than one chip is used, care must be taken to make sure that the chips are piggy-backed in the right order or the program will not function properl This is not too difficult because the chips are numbered in the proper order (DMII-CD2234ML and CD2235, for example). Just make sure the chips are stacked in assending order and everything will work fine. (See figure 2 for more detail.)

To prepare the grom chips for installation do the following:

1) Carefully bend pin 14 on all the grom chips with a needlenose pliers. Refer to figure 1 for location of pin 14.
2) Piggy-back all of the chips used making sure the notches on the chips face the same direction and are arranged in the proper order as discribed above. If more than one program is being installed keep the grom chip groups together.
3) Solder all of the pins excep for the pin 14's. Make sure that there are no solder bridges between the pins.
4) Solder the pin 14's for eac program group together. Solder a thin, 6 inches long, to each program group at pin 14. (See figure 2 for detail.)

Installation of the Program Grom Chips

1) Separate the ribbon cable into two pieces, one with 8 wires and the other with 7 wires.
2) Attach the ribbon cable to the remaining 15 pins on the grom stack. The 8 wire piece is attached to pins 1-8 and the 7 wire piece to pins 9-13, 15 and 16.
3) The wires attached to pin 14 are then connected to the switch.
4) Attach a small piece of wire between the center of the switch and pin 29 of the grom extender. (Figure 3).
5) The wires from pins 1-13, 15 and 16 of the grom stack are attached to the grom extender positions indicated in Table A.
6) Recheck all of the connections.
7) Wrap the grom stack and wires

with electrical tape so that it will not short against the motherboard's metal shielding when installed in the console.

8) Install the switch in the console close to the grom port either on top or in the back.

Reassembling the Console

Before reassembling the console, test the programs installed. Reconnect the power supply, keyboard and monitor to the motherboard. Make sure the power supply and keyboard are on a non-conductive surface before applying any power to the console. Turn on the console and try each of the programs installed to make sure everything is working properly. Also check basic and the grom port for proper operation. It may be necessary to reset the console (fctn =) each time a different program is selected. Make sure that the grom stack switch is in the off position before inserting any cartridges into the grom port. If everything is working fine then the console can be reassembled. If a problem occurs recheck all your work.

When reassembling the console make sure that the ribbon cable is bent out of the way so that the grom port can be reinstalled into the top of the console and it does not interfere with the operation of the console. The grom stack should be placed to the left side of the console above the motherboard. Reassemble the console in the reverse order used to disassemble it.

After the console is assembled recheck it again to make sure everything is operating correctly.

If there are any questions about this project please feel free to ask. My address is: 7167 Luana, Allen Park, MI 48101.

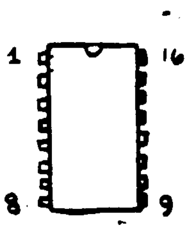
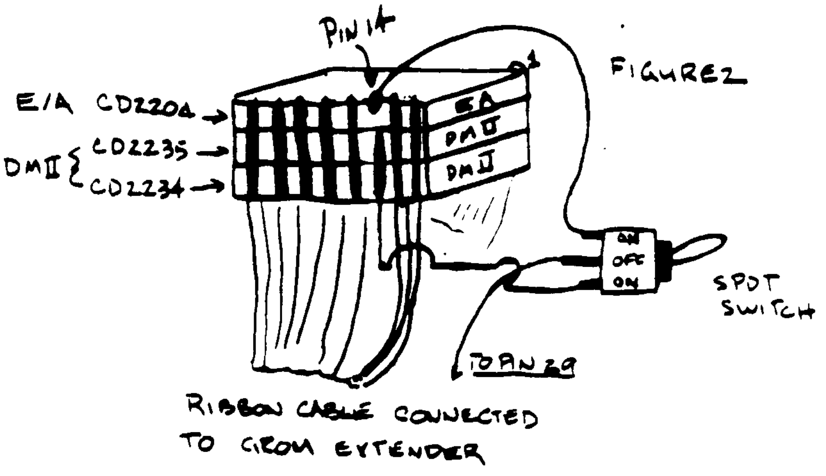


FIGURE 1
GROM CHIP PIN POSITIONS



36	34	32	30	28	26	24	22	20	18	16	14	12	10	8	6	4	2
35	33	31	29	27	25	23	21	19	17	15	13	11	9	7	5	3	1

FIGURE 3
REAR VEIW OF GROM EXTENDER

Grom Extender	Grom Stack
3	1
5	2
7	3
9	4
11	5
13	6
15	7
17	8
19	9
21	10
23	11
25	12
27	13
29	14
31	15
33	16

Table A

Adding a Second RAM Chip

This section describes how I added a second RAM chip by piggybacking it top of the first. However, this makes the chip pile high enough so that t module cover will not close over it. Accordingly, I had to remove a sma section of the top module cover (about 1 by 2 cm.) right at the point where it takes a couple o' right angle turns. This is where the module narrows so that it will fit into the cartridge slot of the console. Since the chips take up some of this space, this "souped-up" Supercart needs to reside in a widgit or other cartridge expander (it even works well in a GK). To do the actual cutting of the module cover, I used an old soldering gun which had a plastic cutting tip but I suppose anything from drills to hot wires could be used also.

The Hitachi HM6264LP-15 is a 28 pin chip of which one pin is not connected, two pins are concerned with power supply (ground and +3-5V input), and 21 pins of which are address and data lines. This leaves 4 pins left over which control the functions of the chip. Pin 27 is the <u>WE</u> or Write Enable pin which determines whether the chip will be written to or read from and is controlled via the wire connected to edge connector 3; if the voltage to this pin is in a high state (+ voltage) then the chip's memory will be available to be read from whereas if it is low (Ø voltage or grounded) then a write to memory is expected. Pin 26 is the CS2 pin which seems to act as a sensor as to whether power is applied or not; if this CS2 pin is at a low (Ø voltage or grounded) state, then none of the chip's memory functions are accessable. This is why it is fed a continuous high voltage state via the LED which is connected to the +5V supply from the console (the left hand F3 hole connects with pin 26). Pin 22 is the <u>OE</u> pin or data bus in and I'm not entirely clear as to its meaning. However this system, if this pin is at a high voltage state, output from the chip disabled and if it is at a low state (Ø voltage or grounded) then read and write functions can be done. The last of the four control pins is pin 2Ø or <u>CS1</u> r chip select pin. When this pin is supplied with a high state (+ voltage) t entire chip pretends that it isn't there (it's "deselected"). When this pin at a low state (Ø voltage or grounded) then it gets the message that it has bee "selected" by the rest of the system to converse with and its functions are enabled. If you look at the inside of a GK or Horizon Ramdisk which both use piggybacked 6264LP-15s, you will find pins 2Ø bent out with individual wires connecting them to the board; this is the way each chip is selected or deselected.

The above paragraph is probably boring and inaccurate but it helps to explain the circuitry necessary to add another RAM chip to the pile. It's relatively simple to piggyback another RAM chip on top of the first; bend in the pins to make a tight fit over the lower chip's pins by molding on a table top, then bend out pins 1, 2, 2Ø, 27, and 28. Then solder the pins from the top chip to the bottom chip being careful not to make any solder bridges between adjacent pins. (In my module, I actually soldered the two together before I installed it on the board.) Pin 1 is ignored. Pins 2, 27, and 28 are connected to the same wires as supply the corresponding pins on the lower chip. If you connected all of the pins of both chips in parallel, you would have both chips doing the exact same thing - clones of each other. How do we give each chip its individuality? This is where the <u>CS1</u> pins (pin 2Ø) become useful. A "pullup" resistor is use to supply + voltage (a high state) to pin 2Ø of the chip not being used which we read in the above paragraph has the effect of making that chip "invisible" to the system. In the absence of such a "pullup" resistor and + voltage source, these pins would tend to "float" down to a Ø voltage state which would cause the system to "select" both chips at once. This would cause the system to read the same address of both chips simultaneously which would result in garbage and a probable crash. In the Supercart board, there is a resistor (R1) which acts such a pullup resistor. In the version described for use in cartridge expanders, this R1 resistor is connected between CS1 (pin 20) and the +5V line from the console. This supplies a high state to deselect the chip. How then is the chip selected to enable it to do its thing? This is the function of the wire connecting pins 20 and 22 (the OE pin). When the OE pin is made a low state (0 voltage) then pin 20 is also made low since the resistor supplies voltage less readily than the direct connection to pin 22 "takes it away". To enable us to use both chips independently then, we could use a switch to connect the OE (pin 22) line to either of the RAM chips pin 20 while having pullup resistors connected to both pins 20 to keep the other chip deselected while the one chip is working.

This is exactly what I did: disconnect any wiring between pins 20 and 22 (to be found on the lower or older chip); next connect 1K resistors (R1 in Figure 3) between pin 20 and the +5V line for both the top and bottom RAM chips; next run wires from pins 20 of both the lower and upper chip to the outer terminals of the SPDT switch; then connect the center terminal of the SPDT switch to the OE pin with another wire (if you're tired of soldering on chip pins by now, you could run this wire to edge connector 2 which is the same line).

I then drilled another 1/4" hole in the front (label) side of the cartridge (somewhere on the left hand side to keep it away from the chips) to install the switch in. If the spring and door of the module cover have been moved to the bottom cover, it makes it easier to insert the modified board back into the module. Again, wrapping any exposed wires helps to prevent short circuits (in one of my earlier efforts, smoke rewarded me when I powered up the Supercart!) I finally used black electrical tape to wrap around the module and cover up the hole I'd made in the top cover. Voila, a manually switchable extra bank of useable memory! Now I can choose between 2 different entry menu screens simply by flipping the switch.

One other potentially useful feature I've found is this: with my previous single banked Supercart, I would more often than not scramble the memory if I removed the cartridge or inserted it with the console power on. (In retrospect, this is because the chip was hardwired to be constantly selected and was subject to transients and "spinal shock" when connected and disconnected.) Now if I "deselect" both RAMs by placing the switch in the center position, I can remove and insert the cartridge even with console power on without losing Supercart contents. To run, however, one or the other of the RAM chips has to be selected.

I hope these comments have been useful to any other "technoklutzes" beside myself out there. If anyone has any corrections or comments to make, I'd be pleased to get them at: Jim McCulloch, 9505 Drake Avenue, Evanston, IL 60203-1107 (CIS ID# 74766,500).

THIS IS INTERESTING

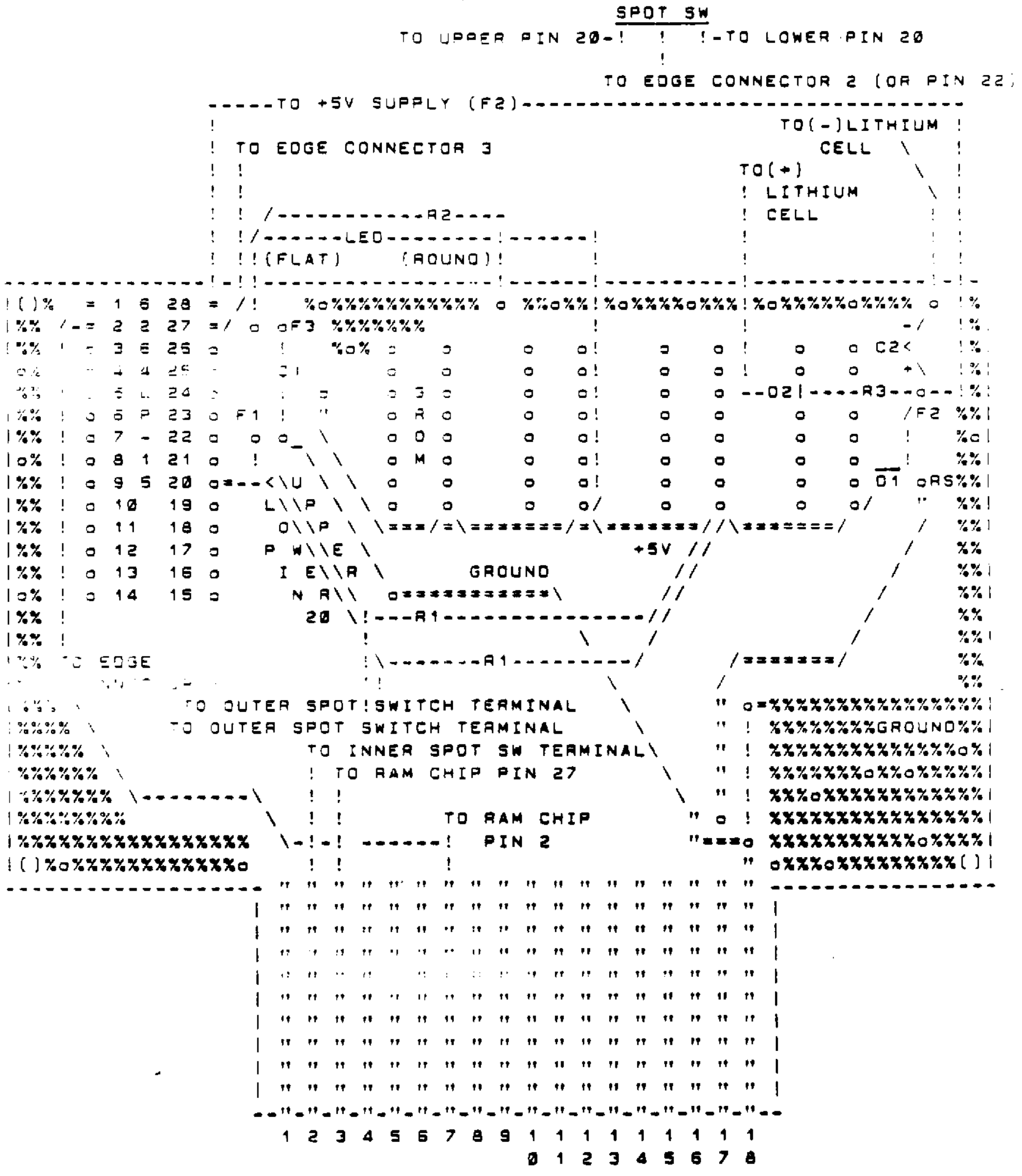


FIGURE 3
(Supercart With Switch Selectable RAM Chips)

DISK DRIVE MODIFICATION INFORMATION
bY PAUL DeMARA, CET 10760 ROSEBROOK RD. RICHMOND B.C. V7A 2R7
WRITE= LOW TO WRITE (BAR ABOVE WORD MEANS LOW LOGIC LEVEL) NORMALY SITS AT +5VOLTS.

2--RESERVED	16-MOTOR ON
--HEAD LOAD	18-DIRECTION IN
6--SEL 4	20-STEP
8--INDEX	22-WRITE DATA
10-SEL 1	24-WRITE GATE
12-SEL 2	26-TRACK GATE
14-SEL 3	28-WRITER PROTECT
	30-READ DATA
	32-SIDE SELECT
	34-READY

These even numbered pins control all functions to the disk drive. Pin number 32 is of interest because it can switch the head to side two electronically on double sided disk drives. Pin 32 is held high logic by the IC on the disk drive and is pulled to ground by the disk controller card when it wants to read side two. If you were to bring pin 32 to ground by adding a switch it would be forced to read side two but the card would not see any change and therefore you could format side two with a directory completely seperate from side A of the disk. This is very helpful when backing up disks or when you want to have two sides with XB loaders on them. You just flip the switch to read side two. It is also possible to modify the circuit to read side two by calling up an unused DSK# eg. DSK3. would read the back side of the disk drive and the controller card would think its reading DSK3. When in fact it is reading side two of one of the other disks. This would make an excellent way to back up SS disks without having to have extra disks and would be very quick as no disk swaping would be neccessary. The side two mod using the ommand DSK3. or DSK4. (3rd party disk controller cards) requires a relay to do he switching and some rewiring of the disk drive is neccessary however the rewards are worth the effort. The two modifications do not in any way effect the disk drives normal operation.

Here is a diagram of the disk drive 32 pin plug:

```
 TOP OF DRIVE    00s001111122222333
                 24p680246802468024
                 ================== <34 PIN EDGE CONNECTOR
                 00c000111112222233   ALL ODD NUMBER PINS ARE CONNECTED
                 13e579135791357913   TO GROUND.
```

To wire the side A side B switch you will need a single pole double throw switch. You then must locate wire number 32 on the ribbon cable that connects the disk drive to the controller card. Cut wire number 32. Then take your switch and take wire 32 from the disk drive and hook it to the center terminal on the switch. Next take wite 32 from the disk controller card and hook it up to one of the outside pins on the switch. The left over outside pin on the switch is then hooked up to wire 33 or any suitable ground on the disk drive. This switch in one position will make the disk drive operate normally. When the switch is flipped the disk drive head switches to side two. One other thing that I found is if you format side two when the disk is formated double sided on side one you will have to first read side one when getting ready to initialize a disk and then flip the switch when the software is ready to format. The reason is the disk manager module will give you an error if you ry to read side 2 before it is reformated. Hope the Modification help all you I users save disk space and save you from hacking up your disks. Anyway if you would like more information or a diagram then feel free to give me a shout during normal hours. Please feel free to copy this info and if you find it helps a donation would be appreciated as I am working on a few other goodies for the TI.

No Special Dress or Posture
Is Required to Telecommunicate.
Come as you are!
GO TI FORUM

A Look At Compuserve

Copyright 1986 Jonathan Zittrain
Compuserve ID 76703,3022

Compuserve's Consumer Information Service is one of the most comprehensive and useful networks available today, especially for the TI user.

Overview

Compuserve is part of CompuServe, Inc., in Columbus, OH. Users from across the country (and lately the world) are able to access Compuserve through local telephone numbers in many metropolitan areas or supplementary networks such as GTE Telenet or Tymnet.

Users are billed by the connect- minute, also based on the time of day. Rates through a standard CompuServe number are $6.25/hour Standard time (6 p.m.-5 a.m.) and $12.75/hour Prime time (8 a.m.-6 p.m.). Weekends and holidays are considered to be Standard time. These charges are based on 300 baud. 1200 baud and 2400 baud costs $12.75/hour Standard and $15.75 Prime.

A Quick Tour

So much is available on Compuserve that it is difficult to choose a representative sampling! In fact, Compuserve itself has an interesting online tour designed especially for new users. Once online, a GO TOUR will show each of Compuserve's main areas.

Compuserve's main structure is in "pages" of text. GO is used to manuever from page to page, and on a particular menu one can choose a selection and be moved to its corresponding page. The very "top" menu, known as page Compuserve -1 or TOP, can be accessed with GO TOP (or even TOP), and looks like this:

```
CompuServe                    TOP

 1 Subscriber Assistance
 2 Find a Topic
 3 Communications/Bulletin Bds.
 4 News/Weather/Sports
 5 Travel
 6 The Electronic MALL/Shopping
 7 Money Matters/Markets
 8 Entertainment/Games
 9 Home/Health/Family
10 Reference/Education
11 Computers/Technology
12 Business/Other Interests
```

The exclamation point (!) is used as a prompt. If you see an !, it means that it's your turn to type something. A prompt in Easyplex, the electronic mail system (which provides user-to-user "mail" as well as a link to MCI Mail), the prompt may be "Easyplex!". At first it may seem that the system is merely excited about the fact that you are using Easyplex and is demonstrating that with the !. It becomes routine soon enough, though.

FIND is another useful command, and functional almost anywhere. We'll be using FIND soon as we look at Compuserve's "forums."

For now, let's take a look at a typical group of text pages. The AP Newswire is a good example:

```
AP Videotex              APV-1

Associated Press News Highlights

 1 Latest News-  7 Entertainment
   Update Hourly
 2 Weather        8 Business News
 3 National       9 Wall Street
 4 Washington    10 Dow Jones Avg
 5 World         11 Feature News
 6 Political     12 History

Enter choice or <CR> for Sports !1

AP Videotex          APV-2647

AP 10/25 22:57 EST V0540

Here is the latest news from The Associated Press:

A stunning new book has been released that is rumored to be a boon to TI
users across the country. Reporters are scrambling for details on this
comprehensive tome, which is said to succeed Dr. Ron Albright, Jr.'s _The
Orphan Chronicles_.

Stay tuned to this wire for news on this story as it breaks!

!
```

<ahem> Well, that's something like what the AP Newswire would report. Also available is the National Weather Service's weather reports, searchable on a given city or state. Aviation reports are surcharged by very detailed, including high-resolution weather maps that are in RLE format (which the TI-99/4A can now support via several third party programs, one of which is public domain).

As you can see, Compuserve is a great way to tap into timely news and other information. The Washington Post and the St. Louis Post-Dispatch are also available in online forms.

Interaction among people from anywhere on Earth is another incredible benefit that Compuserve can provide. Compuserve was first with a CB simulator, a program where people can "talk" in real time. Here, for your viewing pleasure, is a typical slice from a CB conversation. JZ is yours truly:

```
CB II - CB Simulator(sm) v1B(34) Band A
What's your handle! JZ

(channel) users tuned in
  (1)23 (2)7 (11)1 (15)1 (17)9
  (21)2 (24)2 (26)3 (33)15 (34)2
  (35)1 (36)6 (Tlk)32
Select a channel or press
<CR> for more information! 2

Entering open channel...
Key /HELP for assistance

(A2,*Foxy from DC*) hi son
(A2,NIGHTCRAWLER) Hi FOX!!!!!!!
/noecho
(A2,*Foxy from DC*) hi night!
% Echo off
(A2,JZ) Hello there!
(A2,*Foxy from DC*) nice guy...where are you?
(A2,*Foxy from DC*) hi JZ
(A2,*WAYWARD SON*) FOXY M OR F?
(A2,*Beach Baby*) hi jz
(A2,Nice Guy) me too, foxy...gaithersburg
(A2,*Foxy from DC*) son....yes indeed
(A2,NIGHTCRAWLER) Sounds interesting and very impressive,BB
(A2,*WAYWARD SON*) WHERE U FOXY
(A2,JZ) Smile at the camera—
(A2,JZ) this is for a book!
(A2,*Beach Baby*) gee thanks, night
(A2,*Foxy from DC*) <——smiles
(A2,*Foxy from DC*) son...yes I do
(A2,Jc w/HBO) Howdies one et al
(A2,Nice Guy) [[smile]
(A2,*Foxy from DC*) hi JC!

/UST 2

Job  User ID        Nod  Chn  Handle
---- ------------   ---  ---  ------------------
  7  7xxxx,xxxx     HVT   2   *WAYWARD SON*
 20  7xxxx,xxxx     LAK   2   190E
 33  7xxxx,xxx      NYL   2   *LOVERBOY,nyc*
 50  7xxxx,xxxx     NYY   2   NIGHTCRAWLER
 62  7xxxx,xxxx     ORL   2   *Beach Baby*
 88  7xxxx,xxxx     BOO   2   Jc w/HBO
 94  7xxxx,xxxx     DCQ   2   *Foxy from DC*
 96  7xxxx,xxxx     DCI   2   Nice Guy
105  76703,3022     PIS   2   JZ
```

```
(A2,JZ) I'm "taping" here for the next
(A2,JZ) ten seconds to show what CB is
(A2,JZ) like to all those uninformed
(A2,JZ) folks still living in the
(A2,JZ) Stone Age!
(A2,*Foxy from DC*) JZ..that should be enough! heheh
(A2,JZ) <grin>
(A2,JZ) Bye!
```

All CB commands are entered with a slash (/). For example, /EXIT is used to leave the CB area. There are 36 channels available--I happened to be on channel 2. /UST stands for "User STatus," and lists the users on a particular channel. The User ID's of those involved have been changed to x's to protect the innocent! /NOECHO is a nice feature which allows one's typing not be "echoed" back--until <enter> is pressed, at which point the line of text appears as part of the normal CB conversation. As you can see, that conversation can become fairly tangled!

The multi-player gaming areas of Compuserve take the CB concept one step further. "MegaWars" is a game where many users can get together at once, each user being a starship. Since a demonstration of my lowly scout ship being decimated by a much larger dreadnaught would be too graphic and violent for such a G-rated book, I've opted not to include that. The fact that it would also be rather embarrassing is immaterial, of course! The TI-99/4A is perfectly capable of of participating in such areas, however, even when screen protocol is required. Terminal Emulator II is not suggested; instead, Paul Charlton's FAST-TERM or C. Richard Bryant's PTERM99 would be good disk-based terminal emulators to use.

Another more serious application of the multi-user concept is through a Compuserve forum. A forum is an area based on a particular topic, which contains a message base, data libraries (where files and programs are stored for user retrieval) and a conferencing area (a miniature CB simulator).

Here's one page of the results of a FIND FORUM command:

```
!FIND FORUM

CompuServe

 1 ADCIS Forum
   [ ADCIS ]
 2 AI EXPERT Forum
   [ AIE-100 ]
 3 AOPA Forum ($)
   [ AOPA ]
 4 Amiga Forum
   [ AMIGAFORUM ]
 5 Apple User Groups Forum
   [ APPUG ]
 6 Ashton-Tate Forum
   [ ASHFORUM ]
 7 Ask Mr. Fed Forum ($)
```

```
   [ ASKFED ]
 8 Astronomy Forum
   [ ASTROFORUM ]
 9 Atari 16 Bit Forum
   [ ATARI16 ]
10 Atari 8 Bit Forum
   [ ATARI8 ]

MORE !
```

"More" is right! There are quite a few forums (fora?) on Compuserve , and the number increases regularly. Topics range from wine-tasting and health food to (you guessed it) TI computers. In fact, the TI Forum has a very nice "entrance" in page format that can describe itself very well:

```
The TI Forum              TINEWS

================================
    T h e  T I  F o r u m
================================
 1 Introduction
 2 Using The TI Forum
 3 Items of Interest: TI-99/4A
 4 Items of Interest: TI PRO
 5 Enter The TI Forum
 6 What's News
 7 Index of Topics
 8 Feedback/Questions
 9 Masthead/Copyright Info

!1

The TI Forum              TEX-4
```

Find New Friends on

TI Forum

```
     Welcome to the TI Forum! The TI
Forum on CompuServe is an area where
diverse people can meet, trade
information and have discussions around
a central topic of Texas Instruments(tm)
brand computers.
     As a source for information and
programs for TI computers, the TI Forum
is unmatched! Its data libraries
contain surely the best collection of
quality public domain and Fairware
(we'll talk about "fairware" later!)
programs, and topics range from the most
complicated and technical data on the
computers to long-winded discussions on
plugging them in and turning them on!
Whether you're a novice or an expert on
using TI computers, you will find just
the level and amount of information you

MORE !S

desire.
```

Getting to Some Real Benefits . . .

The TI Forum has two major parts: The TI Forum itself, and the accompanying menu pages (recognizable by the TEX-xx page number at the header of each page), which you are reading right now. In order to get the most out of the TI Forum, you should first learn how to use it, and use it efficiently since time is money while on CompuServe. The menu area contains both introductory tutorial-type information on using the FORUM, and also news and items of importance for the TI user. We suggest you first read "Using the TI Forum," selection two from the top menu page of TEX-1. USING THE TI FORUM explains how to become a member (don't panic, it's free!), how to use the Forum, and how to use your time online most effectively. Menu items three and four from the TEX-1 menu allow all users to see the latest news and information for their particular model of TI computer (TI-99/4A and TI Professional, respectively). Menu choice five lets you actually enter the TI Forum and see what it's all about!

So, if this is your first time on or so, check out "Using the TI Forum" and you can then proceed to the FORUM itself if you like. Comments and questions are always welcome and appreciated; the TI Forum Coordinators and members alike are almost always ready to lend a helping hand!

The TI Forum is not affiliated with Texas Instruments, Inc., in any way.

If you have any questions or comments, you can use our "Feedback/Questions" area, option 8 from the main TINEWS menu, TEX-1. You will get a response via Easyplex, usually within 24 hours.

USING THE TINEWS AREA

The TINEWS area will be updated constantly to provide the latest news, help, and information about the TI Forum and the TI world in general. You can check the "New in TINEWS" area occasionally to see information about the latest features.

Now is a good time to introduce some commands that are very useful in navigating around the whole Compuserve page structure.

Keep these on hand, and it's hard to become lost!

Navigation Command Summary

GO xxx - GO to a certain page number. If you have to break off in the middle of a section, you can write down the last page number you saw in the upper right-hand corner and just hit GO xxx, where xxx is that number, to return. For example, GO TEX-200, or just a GO 200 if you are already in the TEX area will take you to the TI Forum. GO TEX-11 (or GO 11 from within TEX) will take you to the "Using the TI Forum" section of TINEWS.

T - Returns you to the TOP page, usually called TOP, but this is a settable page through OPTIONS. Type GO OPTIONS for more information.

M - Previous menu. If you are reading an article or buried under a few menus, this will let you climb out! It takes you to the latest menu you accessed in TINEWS.

S - This is helpful! It stands for SCROLL and will display a series of connected pages (like articles) continuously without making you hit <ENTER> after each page. Great if you are a fast reader or are trying to capture a certain section. Just hit S at the prompt where you are asked to hit <ENTER>, or S x, where x is a menu choice you would like to have scrolled.

N - Proceed to the NEXT menu choice.

P - Proceed to the PREVIOUS menu choice.
B - Go BACK one page. F - Go FORWARD one page. SET WID xx - Set your screen width
to xx. Set to 80 for buffer capture; 40 for regular /4A screen.

The Tour Finale: The Forum Itself

Now, a quick glimpse inside the TI Forum! Signing up for the TI Forum, as with most public forums, means only supplying your name. Some forums accept handles, such as the CB Forum, but the TI Forum prefers full name. Here's the TI Forum main menu:

```
The TI Forum (sm)

FUNCTIONS
 1 (L)  Leave a Message
 2 (R)  Read Messages
 3 (CO) Conference Mode
 4 (DL) Data Libraries
 5 (B)  Bulletins
 6 (MD) Member Directory
 7 (OP) User Options
 8 (IN) Instructions

Enter choice !
```

The conference mode, as mentioned before, is really just a miniature CB for TI users. There is a weekly conference scheduled where the regulars can get together, as well as special events. Craig Miller, Laura Burns & John Koloen (of MICROpendium), and Scott Adams are a few of the people that the forum has been lucky enough to have for an evening in CO. Transcripts of many CO's have been permanently archived in the TINEWS area.

The Member Directory is an area where users can list their own interests and find other users with similar interests based on a keyword search.

Let's go through the process of leaving a particular message. Many messages are left to "ALL", which is the public at large, and replies are then added to create a message "thread." Once the commands are known, it's very easy and convenient to navigate through the message base, reading messages and their replies. A Compuserve forum has, in my opinion, by far the best message base structure of any network or BBS.

We type "L" to leave a message:

```
Enter choice !L
To: John Doe 76543,210
Subject: Charges

Enter /EX to exit EDIT

[ New message ready ]
John -
.
.     You must have misread your
booklet! CompuServe standard
charges are $6/hour, not $0.06/hour!
Sorry for the two hundred dollar
bill you ran up before you realized
that.
.
.     If there's anything else I can
do to be of assistance, pleez let
me know!
.
....JZ
/t
/l/pleez/$
do to be of assistance, pleez let
/c/pleez/please/$
do to be of assistance, please let
/ex
```

```
LEAVE ACTIONS
 1 (S)  Store the Message
 2 (SU) Store Unformatted
 3 (C)  Continue Entering Text
 4 (A)  Abort the Leave Function

Enter choice !SP

SUBTOPIC # REQUIRED

 0 General/Haalp!
 1 TI News & Views
 2 Hardware/Repair
 3 Programming
 4 BBS Help/Views
 5 Tips & Tricks
 6 The TI Trader
 7 The TI Writer (WP)
 8 TI Professional

Enter choice : 0

Message # 94672 Stored
```

When prompted for "To:", if the recipient's User ID is known, it should be appended to the name. This will flag the message for that user when the user next enters the forum with a warning of "You have a message waiting:". Messages last a few days, then "scroll off" as new messages are left. The size of forum message bases vary, from 500 to over 2000 message capacities.

The "/t" moved the pointer in the file to the top, the "/l/pleeze/" located the word "pleeze," and "/c/pleeze/please/" fixed the spelling. EDIT is not difficult to use, although it's not TI-Writer.

Messages can be stored publicly, or with the "P" appended to the <S>tore command, privately. <MA>il is also available as a storing option, moving the message into the Easyplex mail system where it will be more permanent (avoiding the few-day scroll-off), although only the recipient can then see the message.

Now let's read the message by scanning for messages addressed to John Doe. Again we are at the main forum menu:

```
Enter choice !R

READ MESSAGES
 1 (RF) Forward
 2 (RR) Reverse
 3 (RT) Threads
 4 (RS) Search
 5 (RM) Marked
 6 (RI) Individual
 7 (QS) Quick Scan
Enter choice !RS
```

Where Flocks of 99'ers Abound . . .

```
Search Field Menu
(F) From
(S) Subject
(T) To

Search field: T
Search string: Doe
Forum messages:  93952 to  94672
Start at what message number
(N for new to you): 0

#: 94672 (P) S0/General/Haalp!
    28-Oct-86  22:23:18
Sb: Charges
Fm: Jonathan Zittrain 76703,3022
To: John Doe 76543,210

John -

    You must have misread your booklet!
CompuServe standard charges are
$6/hour, not $0.06/hour! Sorry for the
two hundred dollar bill you ran up
before you realized that.

    If there's anything else I can do
to be of assistance, please let me
know!

...JZ
```

And there it is! When Mr. Doe retrieves his message, "(X)" will appear after his name in the "To:" field to show that he has indeed received it. That applies to all messages with a specific recipient. Remember to enter the ID correctly, as well, since the system will not correct for typo's there.

Archived messages of special note are also stored in the TINEWS area in page format. The Great Track Copier Debate, Which Disk Drive is Best?, and TI-Writer Tips & Tricks are just a few titles of messages and message threads that have been saved for posterity.

The Data Libraries contain file after file (all submitted by users; the time it takes to transfer a file to CompuServe is now free of charge--you don't have to pay to give!) of programs, tutorials, and other gems in a multitude of TI-99/4A languages. Companies often store press releases in the DL's as well. TINEWS has plenty of help with file transfer (this is probably the most difficult process for the novice telecommunicator to learn), and a simple BROwse command usually serves to get one started looking through the list of available files.

Conclusions

So, in a nutshell, that's a bird's eye view of CompuServe. I am convinced that it is the finest, most economical service available. For the TI orphan, the TI Forum/TINEWS area is an incredible resource. CompuServe starter kits are available at most computer stores for around $20, including several free hours of online time to get acquainted. A free FEEDBACK area is also available for questions once online.

See you on the network!

A SPECIAL OFFER FROM THE SOURCE®

NOW THAT YOUR PERSONAL COMPUTER CAN TALK TO OTHER COMPUTERS . . .

Talk to The Source!

Now you can join The Source Information Network - A powerful tool to meet both your personal and business needs. The Source offers a broad array of top quality communications, business, personal, and investment services in a network designed to save you time and money online.

And The Source is the only major online information service that gives you a step-by-step guided tour (TUTORIAL) of our most popular services - FREE OF ONLINE CHARGES!

Join The Source today and we'll waive our standard $49.95 registration fee. Your membership can be activated by mailing the attached coupon to:

The Source
P.O. Box 1305
McLean, VA 22102

Sign on TODAY by calling 1-800-336-3366 Toll Free. Just tell the Operator . . .

"I'm taking advantage of your Special $49.95 Fee Waived membership offer, my claim number is ___6450307___.

[] Yes! Sign me up as a member of The Source today with the $49.95 registration fee fully waived.

[] Please send me the SourcePak and Manual at the special reduced price of $12.95 (plus $3.00 postage and handling). Regularly $19.95!

Claim Number 6450307

Bill my registration fee and usage to:

Visa []

MasterCard [] Card number ____________
(must be provided)

American Express [] Exp. Date ____________

Name ______________________________

Address ______________ City ______ State ______ Zip ______

Telephone(Day) ______________ (Evening) ______________

Mothers maiden name (in case ID or password is lost)______________________

I authorize STC to charge all costs incurred on my account as a member of The Source (including usage and monthly membership fee) to the billing service indicated.

Signature______________________________ Date____/____/____

The Source is a registered servicemark of Source Telecomputing Corporation, a subsidiary of The Reader's Digest Association, Inc. ©1986 Source Telecomputing Corp.

Online rates as low as 10¢/minute. $10.00 monthly minimum applies towards usage.

This offer expires 12/31/87.

The GEnie System and TI Roundtable
by Scott Darling, Head Sysop
GEnie ID TIKSOFT

GEnie(tm)(The General Electric Network for Information Exchange) is the newest kid on the block in regards to online information services. In addition to a Texas Instruments RoundTable(tm), there are several other manufacturer specific RoundTables available. GEnie also provides multiplayer game playing scenarios, Computing Today magazine, EAASY Sabre, the American Airlines reservation system, and more....all at the same low base nonprime rate of $5 per hour for 300 or 1200 baud access.

New products soon to appear include more Travel, Shopping, and new Financial related products. There are many more products planned for the future.

Now, why should YOU JOIN GEnie? I mean, isn't it the same as all the others? If this were true, this would be the last line I could write!

Far from it! GEnie is VERY different. The entire structure is unlike any around. Everything in GEnie can be done from Menus or Pages. Each page is numbered and you can navigate easily and fairly fast. GEnie also allows you to go to a specific page and submenu directly from Logon.

YOUR Texas Instruments RoundTable includes a Bulletin Board, Real Time Conference rooms, as well as a Software/Textfile library.

The Bulletin Board function is rather unique. It is based on Topics rather than direct messages to a specific individual. This allows you to follow a specific item or idea along its way.

Structurally, there are specific sections called Categories set up for RoundTable Business, Telecommunicating, Software, Hardware, Basic, Forth, Assembly, Fairware, Gaming, Gram Kracker, TI-PRO, as well as a Newsletter category. These pretty much cover the gamut of things in the TI world. Under each of these categories is where each of the Topics are entered and responded to. Anyone can start a topic, ask questions, and provide answers.

Most of you are used to your local Bulletin Board systems in terms of what to expect and how to react to a message base. GEnie's BBS format differs from your local BBS in certain ways, but as I have done, you too will understand and really appreciate the format.

The RoundTable RealTime Conference is available every Sunday evening for the 4A and the PRO. These are general sessions and are always "free for alls". Whatever questions you may bring with you will most likely be answered during the conferences. This is a Great opportunity to meet and talk with your fellow TI enthusiasts.

The Software Libraries are growing daily. At the time of this writing, they have grown to over 500 files. A lot of the software is Public Domain; the biggest selections are include Fairware and Krackerbox programs. Just about every Fairware program can be found in the TI RoundTable library, including the latest versions. We also have virtually every Gram Kracker program that has been written. I don't have the space to list every program in every library. Suffice to say that there are quite a few and the list is growing. Also, please note that on GEnie, the UPLOADS are FREE! Free uploading is available during Non Prime time(Weekdays between 6 pm and 8 am., all days on Weekends and Holidays).

The file transfer process is also noticeably faster than most other systems. GEnie utilizes their local network nodes for file transfer which results in faster operation than that from the mainframe. Consequently, the numbers just seem to fly by. Nice, especially when you are charged for connect time.

Now the best part about GEnie...the PRICE! There is a one time start up fee of $18.00 to join GEnie, which includes a hardcopy user manual as well as the monthly LiveWire(tm) newsletter. Connect charges are $5.00 per hour for both 300 and 1200 baud during the non prime time hours specified earlier. 2400 baud is also available in over 65 cities throughout the U.S. at an hourly surcharge of $10.00. GEnie is also available during the daytime at a cost of $35.00 per hour for 300 and 1200 baud. The same 2400 baud surcharge also applies during prime time.

Sign up for GEnie is simple and fast. You do not have to order a starter kit. You simply sign up online. Just set up your terminal program for 7 bit, even parity, one stop bit, or 8 bit, one stop bit, no parity; and either 300 or 1200 baud. Also set your terminal to local echo(half duplex). To connect, have your modem dial 1-800-638-8369. After CONNECT, type "HHH" and CARRIAGE RETURN. At the "U#=" prompt, enter "XJM11999,GENIE", followed by a CARRIAGE RETURN. After you are logged on, GEnie will ask you several questions pertaining to your particular system. If you decide to sign up, GEnie will lead you through the electronic signup process, and will ask you for pertinent information. GEnie accepts Visa, MasterCard, and CheckFree. Within two business days following the succesful completion of the Sign up process, a GEnie representative will will call you with your new GEnie User ID#. In a few days following this you will receive your GEnie manual. One last bit of important information. There is NO monthly minimum billing. You only pay for what you use.

The Delphi Network
by Jeff Guide
Delphi Name "TELEDATA"

General Videotex Corporation
3 Blackstone Street
Cambridge, MA 02139

Rates: $7.20/hour, 6:00 PM-7:00 AM, Monday - Friday & all day Saturday and Sunday. $17.40/hour all other times.

Network: Tymnet and Telenet

Start-up costs: Lifetime membership, $49.95 ıcludes DELPHI handbook two hours of free non-prime time useage. DELPHI Starter Kit, $29.95 includes lifetime membership, command card and one hour free non-prime time useage.

DELPHI is a full-service information utility designed for use by the whole family. DELPHI offers electronic mail, teleconferencing and bulletin boards in addition to information services such as weather reports from Accu-weather, news and sports from the Associated Press, business and financial information, trivia tests and movie reviews. It also offers interactive shopping, travel and brokerage services, games and others entertainment features.

Of special interest to Texas Instruments computer users, DELPHI offers the Texas Instruments Information Network (TIIN). The TIIN offers the latest in happenings in the TI world. We offer weekly conferences for the TI 99/4A and the TI Professional computers. Exclusive to the TIIN is a member polling area where you can express your views or start your own poll. The TIIN Shopping Area have the latest products offered in the TI community. Vendors such as Disk Only Software and ASGARD Software offer the latest in hardware and Software for the TI 99/4A computer. Interested in the latest information or public domain material? We have database libraries to supply your needs. Interested in communicating with other users? Have questions on your equipment or know a new technique? Visit our Forum message area and the world is at your keyboard.

As a special offer to those purchasing this book, you can use the TIN on DELPHI tonight! Just follow this procedure: Dial your local Tymnet or Telenet number. When "Please Log In" appears, enter DELPHI. At "Username" enter JOINTI99. At "Password" enter TELEDATA. During this special offer, for $10.00 you will receive a membership account and one hour of non-prime time useage. For $29.95 you will receive a Users Manual, Command Card, membership account and three hours of non-prime time useage.

Join the TIIN and experience a new world of telecommunication.

Editor's Note: I like Delphi; a lot. I think it is a system that deserves better publicity than General Vidtex has given it. Not just the T.I. Information Network (T.I.I.N.) run by Jeff Guide and Dick Ellison (supporting the TI Pro users), but the whole facility is a nice place. Very friendly and easy to navigate. It also has a FREE online encyclopedia (the other systems have one, but have a surcharge for it), and has the only "gateway" service to the massive and complex DIALOG information service. Those two things, alone, are worth the price of admission. But the owners of Delphi appear to be slowly waking up to reality - the reality that the commercial telecommunication business is a competitive one and you have spend some money to compete. The General Videotex people are starting to do just that. If they do, and the word gets out to the world that this is a useful network, Delphi will achieve its rightful choice among the "Big 3" (Compuserve, Source, GEnie). I recommend this network and you can reach me there for mail or on T.I.I.N. with a message. Take advantage now of T.I.I.N.'s generous signup offer - its not offered on any other area of Delphi. Thanks to Jeff Guide for getting us this deal and for pushing General Videotex to get Delphi in the lights.

Find new friends on Delphi . . .

WORD PROCESSING

INSTRUCTIONS AND HINTS
FOR TI-WRITER WORD PROCESSOR
by Dick Altman

IT CAN BE MASTERED! It just takes perseverance and determination and a desire. I have been using it since January 1985 and I don't have it all yet, but I can use it to my immense satisfaction. This came from months of sitting with the large manual in my lap flipping pages back and forth until I had practically memorized the ##% thing! I was at the point where when I had a problem I could say "Oh that is on page 146" or whatever. For instance: this article was done on the TI-WRITER and I now do ALL of my correspondence with it also.

If you received the disk with this article, load it up in TI-WRITER and call it up on the screen so that you can see which commands-and where they were used-to cause the different effects shown in this article. If you received the disk only, then you aren't reading this unless you have already booted it up. It is suggested that you run off a printed copy then reboot this back up so that you can see the commands in use as you read the article. There are comments in the program just below or above the commands that don't show in the printout! This is another 'FREEWARE' item. There is no price set for it. Feel free to pass a copy on to whomever wants it. If it will help only one or two people that are struggling to learn TI-WRITER I will be pleased. If you learn anything from it, and are inclined to fairness, send a few bucks when you can afford it to Dick Altman, 1053 Shrader St., San Francisco, CA 94117. There's no big deal if you don't-only your conscience will know. At least drop me a note and let me know it helped someone.

This is gonna be loo-o-ng, but still much shorter than the 175 page instruction manual!

FIRST RULE: Read the TI-WRITER Quick Reference card and reread it. Of course this means after you read this article. Do all of the operations shown on the card-at least once-even though you might think you will never need that particular one. You will find you have to open up the big manual probably, to accomplish some of the operations. After you have almost 'memorized' the card (literally!) then you will find yourself using it almost exclusively and very seldom having to refer to the cumbersome manual. Personally I think the manual is poorly written.

You will find 3 'windows'-from left to right-to obtain the 80 columns (80 normal characters) width. Each window is 40 columns wide. The first one is from 0 to 40, second one is from 20 to 60, and the third is from 40 to 80. The first thing I do upon booting up TI-WRITER is to set my limits to 37 characters wide. If I take a whole window of 40 characters, it seems to crowd my screen, and I don't like to window back and forth to read my work. I do this by pressing "T" (for TABS), then press ENTER, then placing an "L" on the second dot, and an "R" on the 39th dot, then pressing ENTER again. Now I find my cursor blinking at me from line #0001. Here is where I tell the printer what margins I want it to print my work within. It's also at this point that I select condensed type because I like it better than the normal size type, and I can get 132 characters per line if I wish. It just looks better in my opinion. I normally do this on line 0002 because I used 0001 to set up the formatting (margins, etc.) commands to the printer.

> So, on line 0001 I put in the following 'dot' command (a dot command is merely starting with a period): .LM 20;RM 120;FI;AD (AND END ALL DOT COMMANDS WITH A 'carriage return'). The semicolons are necessary, and the spaces, just as I listed it here. I'll do it again: .LM 20;RM 120;FI;AD(c/r). You of course don't put in the line number 0001. That is already there.
>
> That tells the printer to set the Left Margin at 20, the Right Margin at 120, then Fill each line, and Adust (justify) the right margin. The 'FILL' command tells the program to put in as many whole words on a line, within your predetermined margins, as possible. The 'ADJUST' tells it to add extra blanks between words to cause the even right margin as this article has.

I changed the margin settings on the last two paragraphs just to show you that you can enter your 'commands' just about anywhere within your work!

Just pressing ENTER will normally automatically put in the 'carriage return' symbol, but sometimes it doesn't. It depen what you were doing last. In that case, use Control and 8 to put in a carriage return.

On line 0002 I put in a 'Control' command thusly: Control U Shift 0 Control U. Neither a 'dot' at the beginning, nor a 'carriage return' at the end is necessary. This command throws the printer into 'condensed' type. Neither of these two line numbers will be printed on paper. They are merely formatting commands. Most of the 'Control' commands are listed at the bottom of this article.

Then if I want to center a title (or date) or some other heading at the top of my article, on line 0003 I put in another dot command like this: .CE (remember a carriage return is required at the end of all dot commands). If my title is say three lines of type, then make that dot command thusly: .CE3(c/r) otherwise it will 'center' only one line. The centering command at the top of this article was '.CE5' because of the blank line in it. The lines you wish centered have to immediately follow the centering command.

The automatic page length is 66 lines. This gives you about six blank lines at the top and bottom of your page, and only fifty some actual lines of type. You can, with a dot command change your page length with this: '.PL ##' as I did in line 0002 of this article. (Not enough room in 0001)

Then you start typing your article, letter, whatever. If you wish each paragraph to be indented, it takes another dot command of: .IN(number). If, as in my suggested margin settings of .LM 20;RM 120, you wished to indent each paragraph five spaces, the command would be: .IN 25 because the counting starts at zero or left edge of the paper. If you include the indent command with others in line 0001, the semicolon replaces all but the first dot, thus .LM 20;RM 120;<u>IN 25</u>. You may put more than one dot command on one line, or the Control commands, but never both of them on the same line.

The <u>fun</u> part of a word processor is the capability of inserting or deleting a word or an entire phrase without having to retype the entire page or article. Another fun thing is the ability to move a sentence or an entire paragraph to ar her place in your work. This is all done very simply. Just place your cursor in the last space before where you wish to t another word and press the FCTN key and the number 2. This causes everything beyond your cursor to move down one line. type in your new word or sentence and after the space at the end of it press the Control and the 2 (just once) and ever will [illegible] back up to your cursor. If you are near the beginning of a long paragraph it takes a little longer (a couple or three seconds) to reformat the paragraph, than it does if you are near the bottom of that same paragraph-DON'T GET IMPATIENT AND HIT THE KEYS AGAIN, JUST WAIT A COUPLE OF SECONDS!

To move let's say paragraph #10 into the #3 spot is just as easy. First look at paragraph #10 and make a note (mental??) of the line numbers on the first and last line. Function and zero shows the line numbers or moves them off the screen. Suppose they were 0076 and 0093. Then determine what line number you wish it to be after. Let's suppose it was 0023. Then with FCTN 9 go to the 'command' line, type M (for Move) and hit ENTER. Then type in 0076 0093 0023 and hit ENTER again. Look at those numbers and read the instructions on the Quick Reference Card for MOVE.

On most dot matrix printers, there are two different commands to make neat printing. They are called 'emphasized' and 'double strike'. You can't use (on <u>my</u> printer at least) the emphasized method while in condensed size of type. But I can use double strike. The difference is basically this. Both commands print each letter twice, but in two different ways. One of them (emphasized) moves the head slightly to the right so that each letter is a little thicker. Double strike just prints the line twice. I think emphasized is slightly faster than double strike, but I've never timed either of them. Since I use condensed printing almost exclusively, and can't use emphasized, I don't worry about it. Incidentally, you may enter these commands throughout your article. You just have to have them begin at the left margin of your work. As long as you ber dot commands with a period, and the control commands with Control U (and end dot commands with a carriage return, and c rol commands with Control U and/or a capital letter) you'll be O.K. Only <u>this</u> paragraph was using 'double strike', look at the difference.

An interesting fact about most printers is that it not only inserts unobtrusive spaces here and there to <u>ADJUST</u> each l to the predetermined right margin. IT PRINTS EVERY OTHER LINE FROM THE RIGHT TO THE LEFT while doing all that FILLI ADJUSTING. It will also correctly number your pages if you give it the FO command, which is another dot command.

I find once in awhile, some one command (never the same one twice) seems to falter. Just redo it. Sometimes I think some command must be there that is invisible (this is possible!) so when you run into an unexplainable problem, go back to your formatting command line(s)-which are usually lines 0001 and 0002-put the cursor at the end of each of your commands then press FCTN and 1 and hold them for a couple of seconds to delete any possible typing errors that placed some sort of 'hidden' command in that line.

Another good command to learn is the 'OOOPS' command. Merely Control and the figure one. This eliminates only your last change just now typed in, and returns your work to its former self (hopefully!).

Another good habit to get yourself into, is 'SAVING' your work every few minutes (or every few pages). Power glitches do occur from any power company. Either surges, or stumbles. Sometimes just an electric motor in your home (refrigerator, etc.) kicking in will cause a momentary change in the power supplied to your computer (you've seen your lights flicker). If you save your work every once in awhile, you someday will be glad you were in the habit. Especially if you have just put in to the word processor a 20,000 word story. The power glitch could cause you to lose it all! If you have been saving it on a disk, when that glitch occurs you will have all but a small part of it saved. When you save something to a disk, then come back to that same disk and save something else with the same name, it replaces the first item with the second. It does not become two seperate items on the disk. Of course, if you are really a worry-wart, you will do the saving on two disks, alternating back and forth, just in case that glitch comes while you are in the act of saving your work.

When you wish to reload a file from a disk back into the word processor, it's EASY! When you first bring up the word processor in the Editor mode, you are automatically in the command line. Just type LF (for Load File) and hit ENTER, then type in DSK1.(and the name you gave it) then hit ENTER again and wait a few seconds for the work to be loaded into your computer from the disk.

If you want a rough draft of your work on paper (I find it easier to proof than on the screen) just remove your commands for double strike or emphasizing to conserve your printer ribbon. It will not be so easy to read, unless your ribbon is new, but it will be done faster, as well as not using up ribbon ink unnecessarily.

In the book you will find two methods of going to the disk, then to your printer. Printing should be done from the disk, not from the computer. You will find a command of 'Print File'. That's not the one I use! The one I have become accustomed to using may take a few seconds longer, but it is the one I learned first, and I have just stuck with it. It is as follows. After I have finished typing my letter or whatever, return to the command line with FCTN 9, there type a Q (for Quit) hit ENTER, then S (for Save) and ENTER, then DSK1.TERRY or whatever name I want to give the file instead of TERRY, then ENTER. I usually use a short two or three character name. I have even been known to use #1, or #2, or something like that (the file name cannot be more than 10 characters long, and you can't have any spaces in a file name). Then, after the work goes from the computer to the disk, you can either print it now or sometime next week. The command to go to the printer at this point is like this: Q (for Quit) ENTER, then E (for Exit) and ENTER again. This takes you back to the master menu. This time, you select #2, or THE FORMATTER. After it comes up, you have to type in DSK1.(filename) and hit ENTER. Then you have to type in the command telling it to go from the disk to the printer, instead of to the screen. (With the use of DISKO or some such assembly language repair program, you can insert the command to your printer so that it is a default just like all the other selections on the screen. It is in 'EDITA1' of your TI-WRITER disk.) Without knowing what kind of printer you have, I can't give exactly the correct command here, but it will be something like this: PIO or RS232.BA=4800.LF, then you will have five more choices, mostly for which you will just press ENTER for each of them. Perhaps you might wish more than one copy, so on the correct one you would punch in that number. Be sure your printer is turned 'on' before hitting the last ENTER.(the one that says "PAUSE AT END OF THE PAGE?) because you will be printing immediately.

For your purposes (manuscript writing) you will want it double spaced. That is simply a dot command of '.LS 2' (LS for Line

Spacing of course!) and if you want it triple spaced, just change the 2 to a 3. Or of course use it for a rough draft or some

such. I'm mostly just rambling here, to give this particular paragraph some length, so that you can see double spacing at

work. I can't seem to think of anything else to say, so I will just end it here.

I find once in awhile, some one command (never the same one twice) seems to falter. Just redo it. sometimes I think some command must be there that is invisible (this is possible!) so when you run into an unexplainable problem, go back to your formatting command line(s)-which are usually lines 0001 and 0002-put the cursor at the end of each of your commands then press FCTN and 1 and hold them for a couple of seconds to delete any possible typing errors that placed some sort of 'hidden' command in that line.

Another good command to learn is the 'OOOPS' command. Merely Control and the figure one. This eliminates only your last change just now typed in, and returns your work to its former self (hopefully!).

Another good habit to get yourself into, is 'SAVING' your work every few minutes (or every few pages). Power glitches do occur from any power company. Either surges, or stumbles. Sometimes just an electric motor in your home (refrigerator, etc.) kicking in will cause a momentary change in the power supplied to your computer (you've seen your lights flicker). If you save your work every once in awhile, you someday will be glad you were in the habit. Especially if you have just put in to the word processor a 20,000 word story. The power glitch could cause you to lose it all! If you have been saving it on a disk, when that glitch occurs you will have all but a small part of it saved. When you save something to a disk, then come back to that same disk and save something else with the same name, it replaces the first item with the second. It does not become two seperate items on the disk. Of course, if you are really a worry-wart, you will do the saving on two disks, alternating back and forth, just in case that glitch comes while you are in the act of saving your work.

When you wish to reload a file from a disk back into the word processor, it's EASY! When you first bring up the word processor in the Editor mode, you are automatically in the command line. Just type LF (for Load File) and hit ENTER, then type in DSK1.(and the name you gave it) then hit ENTER again and wait a few seconds for the work to be loaded into your computer from the disk.

If you want a rough draft of your work on paper (I find it easier to proof than on the screen) just remove your commands for double strike or emphasizing to conserve your printer ribbon. It will not be so easy to read, unless your ribbon is new, but it will be done faster, as well as not using up ribbon ink unnecessarily.

In the book you will find two methods of going to the disk, then to your printer. Printing should be done from the disk, not from the computer. You will find a command of 'Print File'. That's not the one I use! The one I have become accustomed to using may take a few seconds longer, but it is the one I learned first, and I have just stuck with it. It is as follows. After I have finished typing my letter or whatever, return to the command line with FCTN 9, there type a Q (for Quit) hit ENTER, then S (for Save) and ENTER, then DSK1.TERRY or whatever name I want to give the file instead of TERRY, then ENTER. I usually use a short two or three character name. I have even been known to use #1, or #2, or something like that (the file name cannot be more than 10 characters long, and you can't have any spaces in a file name). Then, after the work goes from the computer to the disk, you can either print it now or sometime next week. The command to go to the printer at this point is like this: Q (for Quit) ENTER, then E (for Exit) and ENTER again. This takes you back to the master menu. This time, you select #2, or THE FORMATTER. After it comes up, you have to type in DSK1.(filename) and hit ENTER. Then you have to type in the command telling it to go from the disk to the printer, instead of to the screen. (With the use of DISKO or some such assembly language repair program, you can insert the command to your printer so that it is a default just like all the other selections on the screen. It is in 'EDITA1' of your TI-WRITER disk.) Without knowing what kind of printer you have, I can't give exactly the correct command here, but it will be something like this: PIO or RS232.BA=4800.LF, then you will have five more choices, mostly for which you will just press ENTER for each of them. Perhaps you might wish more than one copy, so on the correct one you would punch in that number. Be sure your printer is turned 'on' before hitting the last ENTER,(the one that says 'PAUSE AT END OF THE PAGE?) because you will be printing immediately.

For your purposes (manuscript writing) you will want it double spaced. That is simply a dot command of '.LS 2' (LS for Line Spacing of course!) and if you want it triple spaced, just change the 2 to a 3. Or of course use it for a rough draft or some such. I'm mostly just rambling here, to give this particular paragraph some length, so that you can see double spacing at work. I can't seem to think of anything else to say, so I will just end it here.

There are many, many more commands available, such as merging either parts of two different files, or merging a whole e into the middle of another, or putting in headers at the tops of every page, and footers at the bottom, all automat . Such things as page numbers, or requirements for manuscripts, etc., but those can be found as you need em.

The word processor does have a capacity beyond which you have to save your work to disk, and start with a clean slate. It is approximately 20,000 characters including blanks. I have only run into it when transferring a long story to disk. I was entering a 10,000 word story, and I got 'MEMORY FULL-SAVE OR PURGE' flashing at me at the top or command line after about 4,000 words (I wish it would ring a bell or something). At that point 'save' your work and retire that file name. Perhaps in this article I am writing for you I will reach that point again. Right now I am typing on line number 466. I think it was at about line 400 plus (but I was using 80 column width that time for a special project, I think) that the MEMORY FULL thing happened to me. You will just have to tri and error it for your job! Of course, the length canNOT be judged just by the line numbers on the left side of your screen. Think about whether you are using only one window, or two, or the maximum of three. I am using just one window while I do this work, as I explained earlier, so that will make my capacity come much farther down the line numbers than if I were using all three windows! 80 characters (or columns) wide, instead of the 37 I am using. If and when the MEMORY FULL bit happens to you, remember that when you save it this time to a disk, then for pete's sake don't save the next time to the same file name! In other words, my name for this file at the moment is TI-WRITER. If I need to make a new file, it will become TI-WRITER2.

The little 25 page booklet from Dr. Bill Browning is very good, don't ignore it when you are trying to learn the TI-WRITER word processor. 7541 Jersey Avenue North, Brooklyn Park, MN 55428. Price just $6.50 and worth every penny.

There is also available in 'FREEWARE' circles an excellent disk called "TK-WRITER" which was done by TOM KNIGHT, thus the 'TK'. It replaces the need for a cartridge to have TI-WRITER word processing capabilities. As far as I can tell, it does exactly the same things the cartridge does, except for Show Directory-which is inconsequential, and won't go direct from the Editor stage to the Formatting stage. You can probably find it in the same library you obtained this disk from.

The command for the underscore is merely the ampersand (Shift 7) and it can be used anywhere. Note even in the middle a word 'cannot'. If you want to underline more_than_one_word you have to connect them with what is called a caret. It is the 6, or Shift 6. If you wish, the AMPERSAND can be printed in your work, but not the caret. Merely type in two ampersands and only one of them will be printed! & & &

Believe me, all of this will become easy and second nature to a good typist in a very short time! But if you don't use it for a month or two, you will find yourself going back and back and back to the big book!

Thanks so much to Dr. Guy Romano for his assistance in writing this article. Plus his enormous patience with my dumb questions over the past few months while I was learning the TI-WRITER. Also to Hal White and to Larry Rosenberg for their invaluable assistance. And to Terry & Paul Anderman for their desire to have word processing capabilities, which forced me to finally write this that had been nagging at me so long.

!!!!!!!!!!

CONTROL COMMANDS

****

ASCII CODES	FUNCTION	FORMAT
0	Terminate Tabulation	CTRL U, SHIFT 2, CTRL U
7	Sound the buzzer	CTRL U, SHIFT G, CTRL U
8	Backspace	CTRL U, SHIFT H, CTRL U
9	Horizontal tabulation	CTRL U, SHIFT I, CTRL U
10	Line feed	CTRL U, SHIFT J, CTR
11	Vertical tabulation	CTRL U, SHIFT K, CTRL

12	Form feed	CTRL U, SHIFT L, CTRL U
13	Carriage return	CTRL U, SHIFT M, CTRL U
14	Print enlarged characters	CTRL U, SHIFT N, CTRL U
15	Print condensed characters	CTRL U, SHIFT O, CTRL U
17	Select printer	CTRL U, SHIFT Q, CTRL U
18	Turn off condensed printing	CTRL U, SHIFT R, CTRL U
19	Disable printer	CTRL U, SHIFT S, CTRL U
20	Turn off enlarged printing	CTRL U, SHIFT T, CTRL U
27	Escape	CTRL U, FCTN R, CTRL U
27:48	Set line spacing 8 per inch	CTRL U, FCTN R, CTRL U, 0
27:50	Set line spacing 6 per inch	CTRL U, FCTN R, CTRL U, 2
27:51	Set line spacing n/216 per inch	CTRL U, FCTN R, CTRL U, 3
27:52	Turn Italic Character set on	CTRL U, FCTN R, CTRL U, 4
27:53	Turn Italic Character set off	CTRL U, FCTN R, CTRL U, 5
27:56	Disable paper-end detector	CTRL U, FCTN R, CTRL U, 8
27:57	Select paper-end detector	CTRL U, FCTN R, CTRL U, 9
27:65	Set line spacing(1/72 to 85/72 inch)	CTRL U, FCTN R, CTRL U, A.
27:66	Set up 8 vertical tab pos.	CTRL U, FCTN R, CTRL U, B
27:67	Set form length up to 127 lines	CTRL U, FCTN R, CTRL U, C.
27:68	Set up to 12 horizontal tab positions	CTRL U, FCTN R, CTRL U, D
27:69	Turn on emphasized printing	CTRL U, FCTN R, CTRL U, E
27:70	Turn off emphasized printing	CTRL U, FCTN R, CTRL U, F
27:71	Turn on double printing	CTRL U, FCTN R, CTRL U, G
27:72	Turn off double printing	CTRL U, FCTN R, CTRL U, H
27:75	Turn on normal density graphic printing	CTRL U, FCTN R, CTRL U, K
27:76	Turn on dual density graphic printing	CTRL U, FCTN R, CTRL U, L
27:77	Turn Elite mode ON	CTRL U, FCTN R, CTRL U, M
27:78	Set skip-over perforation	CTRL U, FCTN R, CTRL U, N
27:79	Release skip-over perforations	CTRL U, FCTN R, CTRL U, O
27:80	Turn Elite mode OFF	CTRL U, FCTN R, CTRL U, P
27:81	Set a column width	CTRL U, FCTN R, CTRL U, Q
27:82	Select 1 of 8 int'l char.sets	CTRL U, FCTN R, CTRL U, R

TI-WRITER UNDERLINING

Those of you who use TI-Writer and the ampersand for underlinin may find even if your printer has full underlining, you still g "dashed" underlining. I first realized this when I switched from th TI Impact to the Star Micronics Delta 10.

When I first got the new printer, I messed around with the control codes and found the printer had full underlining capabilities. I later used the ampersand in TI-Writer to do some underlining and got the dashed underlining. I then realized the underlining characters used by TI-Writer's ampersand were defined by TI-Writer.

The codes used on the Delta 10 (the Gemini uses the same underline codes) are CHR$(27);CHR$(45);CHR$(1) to turn the underlining feature on and CHR$(27);CHR$(45);CHR$(0) to turn the underlining feature off. The command used in TI-Writer was ".TL 91:27,45,1". This command assigned the left square bracket,function R (or ASCII 91) the values needed to turn on true underlining. Another command, ".TL 93:27,45,0", the right square bracket, function T (or ASCII 93) was assigned those values needed to turn off underlining.

Of course, these commands must be used in conjunction with TI Writer's formatter. If your printer has true underlining capabilities and you have been using TI Writer's ampersand command, you may wish to start using the transliterate command to do your underlining.

The TI Writer The Easy Way to Communicate

MAYBE YOU DIDN'T THINK OF DOING THIS WITH YOUR TI-WRITER
By Bruce Larson

ONE DISK DRIVE - If you only own one drive, here's a trick to save wear and tear on you and your equipement. Make a working copy of your TI-Writer disk, with only the EDITA1, EDITA2, FORMA1, and FORMA2 files. This will leave you 271 free sectors to temporarily save material you're working on until you have everything perfected. No more pushing and pulling disks while you go from editor to formatter, back to editor, back to formatter...ad infinitum! By the way this article occupies 14 sectors.

REM STATEMENTS - Want to document a program without adding REM statements? List your program to "DSK1.NAME" instead of "PIO" or "RS232". This creates a Display/Variable 80 file of your progr listing which can be read by TI/Writer. Now you can add commen instructions, etc., that will appear on a printout of this file. A word of caution! Make sure the Display/Variable 80 file name is different than your program name or you might find yourself with a beautifully documented program that won't load!

EXTEND THE USE OF TI-WRITER

By Allen Burt - England

From Northwest Ohio 99'er News, May, 86

TI-WRITER can be used for much more than just producing tters--a substitute for a typewriter. In the last article described how to make use of the CONTROL "U" function in the Text Editor mode. This function can be used to extend the application of the system and to produce integrated documents of words and diagrams. For example, it is easy to show a Histogram (Bar Chart) like Figure 1. This uses the CHR(124) obtained by using "FUNCTION" F AND KEY "A" for the verticles and the underline character CHR(95),FUNCTION "-" and KEY "U".

A useful tip when doing this type of exercise is that if you place the CHR(124)'s in the appropriate locations and wish to continue them downwards from the point indicated by the asterisk - just move the cursor down to the next line and press CONTROL "C" and key "5 - this copies the line above onto that line. When you draw diagrams like this, it is better to insert a number of lines in order to have room to move around.

If you want to include a simple graph within your script, try doing this as shown in Figure 2 below. A more sophisticated graph can be achieved using the above techniques. In the example I found the "COPY" command very useful because having once obtained the required width - I only had to "copy" down the required number of lines using (CONTROL & KEY "*). Remember that when you place a special set of codes at the start of the line, the space they occupy will not be recognized by the printer. That is,the printed ne will commence at the location of the first special code. This can place the numbers used in the graph in the wrong place. You have to enter your special codes at the point you wish the following characters to print. Thus, what you see on the screen is not necessarily what you will get on the printout.

TI-WRITER can be used to draw graphs as Figure 3 illustrates. The horizontal lines are achieved by setting the printer into an underline mode (CHR$27;CHR$(45);CHR$(I)). The line spacing is set to 7/72" (CHR$(27);"A"CHR$(7) - This approximates to 1/10". If a CARRIAGE RETURN is placed at the point where the line should finish, the printer will draw a line to that point. The verticle lines are drawn by using CHR$(124) - Function "A". As the printer normally prints at 10 characters to the inch, this will produce a grid of roughly 1/10" squares.

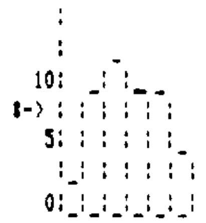
There are 2 points to watch using this procedure:

1. If you do not want the underlining to start at the beginning of the printer line, the underline code must be placed at the start of each line and cancelled at the end of each line before the carriage return. There is another means of achieving this and that is to set the left hand margin to the required position (on GEMINI printers this is CHR$(27);"M"CHR$(N) - n being the column to start printing. THIS CAN ONLY BE DONE USING THE PRINTER CODES, NOT BY TTING TI-WRITER'S TABS.

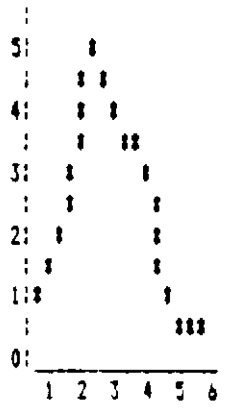
The second point is that many printers do not align the characters in a bidirectional mode. YOU ARE ADVISED IN THE TI-WRITER MANUAL THAT FOR TABULATION, IT IS ADVISABLE TO SET THE PRINTER TO A UNI-DIRECTIONAL PRINTING MODE.

Figure 4 illustrates how a line will appear on the 4A screen.

Figure 1



F i g u r e 2



USING TI-WRITER TO DRAW A GRAPH

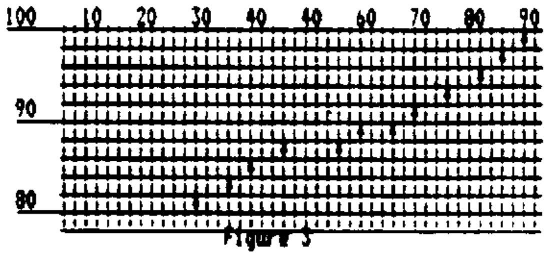


Figure 3

Actual point where printing will start.

figure 4

CHARACTER GRAPHICS WITH TI-WRITER

by Rod Cook
OH-MI-TI

Graphic characters can be defined in TI-WRITER and printed to the printer by using a combination of the transliterate command and the graphics control codes of the printer.

The transliterate command has the format:

```
.TL n:char#,char#,·····,char#
```

where n is the special character number and char# is the decimal number to be transmitted to the printer. For example, in the following command:

```
.TL 0:46,84,76
```

anytime the special chartacter "ȯ" is encountered in the text, the FORMATTER will transmit the numbers 46, 84 and 76 which in this case happen to be a period followed by a capital T and L. The value of char# can have any value between 0 and 255 although ASCII values only go up to 127.

The transliterate command will define the graphics for the character to be printed. One transliterate command per character will be required. Each command will have essentially two parts; the control codes to setup the printer into graphics mode and the data. For the purpose of illustration, the control codes discussed will be for an Epson MX 80. The control code for printer graphics is:

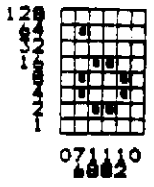
```
<ESC> "K" N1 N2
```

where <ESC> is the escape character, number 27 and "K" is number 75. N1 and N2 are numbers that are used to specify how many data numbers follow. This control code puts the printer into a graphics mode that prints 480 dots per 8 inches. An 80 character line is also 8 inches long, therefore 480/80 = 6. There are 6 dots per character and it will take six data numbers to specify the character. The control code portion of the transliterate command will look like this:

```
.TL 0:27,75,6,0,···data···
```

where N1 is 6 and N2 is 0 which tells the printer there will be 6 data numbers to follow.

The data numbers tell the printer which of the 8 pins to fire on the printhead for each of the six verticle rows of dots that make up the character. For example the graphics for the special character ȯ are coded as follows:



where in verticle row 1 none of the dots are on so they add up to zero. In verticle row two the dots at 4,8 and 64 are on so they add up to 76. In verticle row three 16 and 2 dots are on so they add up to 18 and so on with the remaining three verticle rows.

So the transliterate command to print the special character ȯ looks like this:

```
.TL 0:27,75,6,0,0,76,18,18,12,0
```

and anytime the special character for zero (shift 2 in the TI-WRITER special character mode) is encountered in the text by the FORMATTER the transliterated values will be sent to the printer which will result in the defined graphics character being printed.

The 6 by 8 grid that was printed above was printed this way. Four graphics characters are needed to build the grid. They are:

┌ │ ‾ ╓

The respective definitions are:

```
.TL 65:27,75,6,0,255,128,128,
     128,128,128
.TL 66:27,75,6,0,255,0,0,0,0,0
.TL 67:27,75,6,0,128,128,128,
     128,128,128
.TL 68:27,75,6,0,255,128,188,
     188,188,128
```

so that anytime ASCII characters 65, 66, 67 or 68 are encountered in the text, the above graphics characters will be printed. ASCII character 65 is an A, 66 is a B, 67 is a C and 68 is a D. So to print the above grid, the following pattern of characters would be needed:

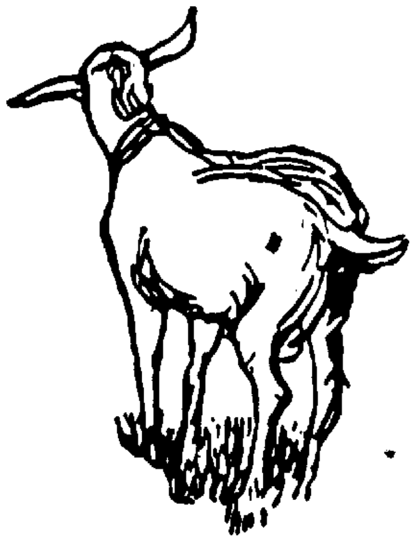
```
128 AAAAAAB
 64 ADAAAAB
 32 AAAAAAB
 16 AADDAAB
  8 ADAADAB
  4 ADAADAB
  2 AADDAAB
  1 AAAAAAB
    CCCCCC
```

Note the numbers will be printed just as they appear because they have not been redefined. Once the graphics have been printed the A thru D will have to be transliterated back their regular character if they are to be used in text. The following commands will do this:

```
.TL 65:65
.TL 66:66
.TL 67:67
.TL 68:68
```

There are some limitations as a result of working within the limits of the FORMATTER.

1. It appears the physical length of the transliterate command can not be greater than one line.
2. It also appears that each character is limited to 6 verticle rows of dots. I have not been able to print a character longer than 6 rows within the FORMATTER.
3. The FORMATTER insists on putting white space on the top and bottom of the page. To print graphics that are continuous from line to line, as is the grid above, requires a line spacing less than that of six lines per inch. To keep the same white space at the bottom of the page and on subsequent pages will require adjusting the line spacing after the graphics to compensate for smaller line spacing of the graphics.



TI Writer Formatter Commands
by Tom Kennedy

Text Dimension commands, as the name implies, move or shape the words in the document (margins, linespacing, right justify, etc.)

.FI	FILL	PUTS AS MANY WORDS ON A LINE AS WILL FIT.
.NF	NO FILL	CANCELS FILL.
.AD	ADJUST	ALIGNS THE TEXT TO THE LEFT AND RIGHT MARGINS.
		(RT. JUSTIFY)
.NA	NO ADJUST	CANCELS ADJUST.
.LM n	LF MARGIN	SETS LEFT MARGIN TO "n".
.RM n	RT MARGIN	SETS RIGHT MARGIN TO "n".
.IN n	INDENT	CREATES AN AUTO-INDENT FROM LEFT MARGIN.
.LS n	LINE SP	SETS LINE SPACING TO "n" LINES.
.PL n	PG LENGTH	DEFINES NUMBER OF LINES TO A PAGE.
.BP	BEGIN PG	DEFINES FIRST LINE OF NEW PAGE.

Internal Format commands control the spacing of characters on a line.

.SP n	SPACE	SIMILAR TO THE TAB FUNCTION.
.CE n	CENTER	CENTERS NEXT "n" LINES BETWEEN MARGINS.

Highlighting commands control functions such as underline or bold and allow you to redefine characters to use them to send CTRL codes to the printer.

^	REQUIRED	JOINS WORDS TOGETHER WHEN REQUIRED TO PREVENT
	SPACE	SPLITTING IN REFORMATING, UNDERLINE, ETC.
&	UNDERLINE	UNDERLINES ALL TEXT FOLLOWING UNTIL NEXT PACE.
@	BOLD	(OVERSTRIKE) RETYPES FOLLOWING TEXT FOUR TIMES.
.TL xx	TRANS-	REASSIGNS ONE CHARACTER TO REPRESENT A NUMBER
	LITERATE	OF CHARACTER VALUES TO SEND CODES TO THE PRINTER.
.CO t	COMMENT	LIKE REM IN BASIC--ALLOWS NOTES THAT DONT PRINT.

Page identification commands print notes in the upper or lower corner of each page, either headers or footers.

.HE t	HEADER	PRINTS TEXT (t) AND PAGE NUMBER AT TOP OF PAGE.
.FO t	FOOTER	PRINTS TEXT (t) AND PAGE NUMBER AT BOTTOM OF PAGE.
.PA	PAGE #	RESETS PAGE NUMBER IN .HE AND .FO

File management commands

.IF f	INCLUDE	MERGES A FILE TO PRINT A DOCUMENT TOO LARGE
	FILE	TO PRINT AS ONE FILE.

Mail Merge option commands are used to supply values to the variables in a letter that has been set up for the mail merge option

.ML f	MAIL LIST	IDENTIFIES VALUE FILE (f) FOR MAIL LIST.
n	VARIABLE	INSERTED IN TEXT AS VARIABLE FOR ASSIGNMENT
		FROM VALUE FILE.
.DP n:t:	DISPLAY	PROMPTS YOU USING TEXT "t" TO ASSIGN.
	PROMPT	TO VARIABLE (*n*).

A HANDY DANDY TI-WRITER USERS REFERENCE GUI

SUBMITTED BY BOB STEPHENS

The following handy TI-WRITER commands are reprinted for the June issue of the 99'er News published by the TI Users Group of Will County, Romeoville, Il. This puts the most used commands on one page for handy access at your computer.

EDITOR COMMAND	FCTN	CTRL	EDITOR COMMAND	FCTN	CRTL	EDITOR COMMAND	FCTN	CTRL
Back tab		T	Ins. Blank line	8	O	Quit	=	
Beginning/line		V	Insert character	2	G	Reformat		2orR
Command/escape	9	C	Last paragraph		6orH	Right arrow	D	D
Delete character	1	F	Left arrow	S	S	Roll down	4	A
Del. end of line		K	Left margin rel.		Y	Roll up	6	B
Delete line	3	N	New page		9orP	Screen color		3
Line #'s(on/off)	0		New paragraph		8orM	Tab	7	I
Down arrow	X	A	Next paragraph		4orJ	Up arrow	E	E
Duplicate line		5	Next window	5		Word tab		7orW
Home cursor		L	Oops!		1orZ	Word wrap/fixed		O

```
Load files: LF (enter) DSK1.FILENAME  (load entire file)
            LF (enter) 3 DSK1.FILENAME  (merges filename with data in memory
                                         after line 3)
            LF (enter) 3 1 10 DSK1.FILENAME  (lines 1 thru 10 of filename a
                                              merged after line 3 in memory.
            LF (enter) 1 10 DSK1.FILENAME  (loads lines 1 thru 10 of filena
================================================================================
            SF (enter) DSK1.FILENAME  (save entire file)
            SF (enter) 1 10 DSK1.FILENAME (save lines 1 thru 10)
================================================================================
Print Files:PF (enter) PIO (prints control characters and line numbers)
            PF (enter) C PIO (prints with no control characters)
            PF (enter) L PIO (prints 74 characters with line numbers)
            PF (enter) F PIO (prints fixed 80 format)
            PF (enter) 1 10 PIO (prints lines 1 thru 10)
NOTE: The above assumes PIO. DSK1.FILENAME, and RS232 are also valid!
      To cancel the print command press FCTN 4.
================================================================================
Delete file:DF (enter) DSK1.FILENAME
================================================================================
Setting Margins and Tabs: (16 tabs maximum)
      L - Left margin      R - Right margin     I - Indent       T - Tab
         Use ENTER to execute or COMMAND/ESCAPE to terminate command.
================================================================================
Recover Edit: RE (enter) Y or N
================================================================================
Line move:  M (enter) 2 6 10  (moves lines 2 thru 6 after line 10)
            M (enter) 2 2 10  (moves line 2 after line 10)
================================================================================
Copy:       same as move except use C instead of M.
================================================================================
Find String: FS (enter) /string/  (will look for string in entire file)
             FS (enter) 1 15 /string/  (will look for string in lines 2 thru 15)
================================================================================
Delete:      D (enter) 10 15 (deletes lines 10 thru 15 in memory)
================================================================================
```

Bit-Image Graphics on Dot-Matrix Printers
by Tom Kennedy

I want to show how you can create your own Basic programs to print Bit-Mapped graphics to Dot Matrix printers. In this case, I use program examples in TI BASIC, and refer to Epson compatible printers, although similar commands apply to various printers.

First, what is Bit-Mapped? When you print a file through your printer, such as a text file, or a Basic program listing, the computer is sending a stream of numbers (Bytes) that represent a predefined code for the various letters, numbers, and symbols we can generate from the keyboard. The printer acts upon built-in programming to convert these bytes to the 7x9 pattern that we think of as "A Character".

When the printer is set to Bit-Image mode, the built-in programming is bypassed, and you must supply the data to fire each of the pins in the print head (the Bits) to "Map" the graphic you want. Considering the number of possible pin positions on a line, with 8 vertical pins per position, this sounds like a arduous task, but there are shortcuts that make it simpler.

In Bit-Image mode, only the top 8 pins of the 9 pin head are used. To fire the pins, you send a byte, in decimal, equal to the Binary value of the arrangement of the pins in the head, as illustrated below:

```
MSB o     M      L
    o     S      S
    *     B      B         So in this case, if the number
    o  =  00101110  =  46  "46" is sent to the printer, the
    *     (Binary)   (Dec) appropriate pins will fire.
    *
    *
LSB o
```

There are two ways to figure the decimal values for the bytes to send. If you are good with Binary/Decimal conversion, or use a suitable calculator, just convert the binary number. Otherwise, a simpler way to look at it is this:

```
     o = 128
     o =  64
     o =  32    Find the numbers that coincide with the pins you
Pin  o =  16    want "on", and add them all up. The result is the
#'s  o =   8    decimal value to send.
     o =   4
     o =   2
     o =   1
```

So now you can create the bytes necessary to print a vertical array of 8 dots, now what? Eight dots do not a picture make. Start at the beginning. Draw a picture, preferably on a fine-grid paper. After you have the picture on graph paper, you break up the graph into 8 squares per row, and try to "frame" the picture in the least number of rows, to save effort. Once you have your rows framed off,

you can asign the bytes needed to create each 1-bit wide column
The procedure for printing the data is to begin by sending
string of bytes necessary to invoke bit-mapped mode, then print
bytes of data, all as one string. Each separate print statement must
be preceded with the set-up string, so it's usually simpler to print
all data for one line all at once.

So, in the example I'm using, I have set off the field into
8-dot rows, so I begin by going down the row, column by column, and
write down the values for the bytes. The first row gives these data
values: (The (#) is the number of times to repeat the value)

(32)0,(4)3,(6)12,(2)15,(2)0,(2)15,(4)48,(2)15,(6)0

To print the first row, you open the printer file. At this point
you also set the printer to 7/72" line spacing, which is height of
the print head. Next, you send the "set string", which puts the
printer in Bit-Mapped mode, and defines how many values will be sent
Now you start a loop tat reads in data values and the repetition
number that prints that data. After all data for one line is read,
you signify this with a "Wild Card" data value, which terminates the
loop. Lastly, you send a "CR" and a "LF", which returns the
carriage.
Susequent rows are printed by repeating the above steps, except
for the opening of the printer.
After all rows are sent, you can reset the printer by sending a
ESC("@").

The following is a BASIC program which demonstrates the abc
procedure. Although written in TI Extended BASIC, the procedure is
the same in any BASIC.

```
 90 REM WRITTEN IN TI EXTENDED BASIC
100 REM THIS EXAMPLE PRINTS A FROG HEAD, IN BIT MAPPED GRAPHICS
110 REM DATA IS READ IN FORM OF: REPETITION,VALUE
120 REM WHICH PRINTS RPT$(CHR$(VALUE),REPETITION)
130 REM
140 REM by Tom Kennedy   December, '85
150 REM    (206) 248-2218
160 REM
170 REM *****INIT VARIABLES (OPTIONAL)**********************
180 P$="PIO.CR.LF" :: E$=CHR$(27):: CR$=CHR$(10)&CHR$(13)::
 SET$=E$&"K"&CHR$(60)&CHR$(0)
190 REM OPEN PRINTER & SET LINE SPACE TO 7/72"*****************
200 OPEN #1:P$ :: PRINT #1:E$&CHR$(49)&CR$
210 REM *********** BEGIN PROGRAM ******************************
220 FOR ROW=1 TO 5 :: PRINT #1:SET$
230 REM ***READ DATA, CHECK FOR END-OF-LINE********************
240 READ NMBR,VALUE :: IF NMBR=99 THEN 270
250 REM *******PRINT DATA FOR ONE LINE*************************
260 FOR BYTE=1 TO NMBR :: PRINT #1:CHR$(VALUE):: NEXT BYTE :: GOTO 240
270 PRINT #1:CR$ :: NEXT ROW
280 REM ********RESET PRINTER (OPTIONAL)***********************
290 PRINT #1:E$&"@"
```

```
300 REM ********************************************************
310 REM DATA VALUES = "REPETITION NUMBER,BYTE VALUE"**************
315 REM ********"99,99" SIGNIFIES END-OF-LINE********************
320 DATA 32,0,4,3,6,12,2,15,2,0,2,15,4,48,2,15,6,0,99,99
330 DATA 24,0,2,3,2,15,2,63,6,255,2,0,4,63,2,255,2,243,2,192,
4,252,2,255,6,0,99,99
340 DATA 20,0,2,15,2,63,6,255,2,252,4,240,6,48,2,240,2,0,8,192,2,
240,2,60,2,15,99,99
350 DATA 16,0,2,3,6,255,2,192,6,0,2,255,10,12,8,48,2,192,2,
195,2,204,2,240,99,99
360 DATA 6,0,4,15,4,63,10,255,8,0,2,192,12,0,2,3,2,
12,2,48,2,192,6,0,99,99
```

Everything I've covered so far is on printing graphics, but there is more to the Bit Image modes. In the above example, I used the ESC("K") to set the mode, but there are actualy eight modes to chose from. The modes are numbered 0 to 7, with 0 (ESC("K")) being the simplest. The following table lists the modes.

MODE	DENSITY	ALTERNATE CODE	DESCRIPTION	HEAD SPEED (in/sec)
0	SINGLE	ESC K	60 DOTS PER INCH 480 DOTS PER LINE	16
1	LOW SPEED DOUBLE	ESC L	120 DOTS PER INCH 960 DOTS PER LINE	8
2	HIGH SPEED DOUBLE	ESC Y	SAME AS MODE 1, BUT FASTER. CONSECUTIVE DOTS NOT PRINTED.	16
3	QUADRUPLE	ESC Z	240 DOTS PER INCH 1920 DOTS PER LINE CONSECUTIVE DOTS NOT PRINTED.	8
4	CRT 1	NONE	80 DOTS PER INCH 640 DOTS PER LINE SAME DENSITY AS EPSON QX-10 COMPUTER	8
5	ONE-TO-ONE (PLOTTER)	NONE	72 DOTS PER INCH 576 DOTS PER LINE EQUAL DENSITY IN BOTH HORIZONTAL & VERTICAL.	12
6	CRT II	NONE	90 DOTS PER INCH 720 DOTS PER LINE	8
7	DUAL DENSITY PLOTTER	NONE	144 DOTS PER INCH 1152 DOTS PER LINE TWICE DENSITY OF MODE 5	3

All modes can be activated with the sequence ESC("*"), foll
by the mode number. Modes 0-3 also have the alternate ESCape value
(K,L,Y,&Z). In this tutorial, I will deal with these four, and in
most cases you will only use the first two.

The difference between mode 0 (ESC("K")) and mode 1 (ESC("L"))
is that in mode 0 one dot is printed spaced at the width of the prin
head pins. In mode 1, a dot is printed every 1/2 pin width, producing
double density.

Mode 2 (ESC("Y")) is the same as mode 1, except twice as fast.
The speed is attained by not printing consecutive dots in a row. This
feature might be a problem in fine detail, such as lettering, but not
when printing a large graphic, such as a screen dump. Mode 3
ESC("Z") is the same as mode 1, but a dot is printed every 1/4 pin
width, and consecutive dots are omitted.

The second part of the set-up string, after the mode values, are
the numbers that define how many columns, or dots, will be printed
per row. Remember, this is not the same as how many characters per
inch might be defined when selecting a font in normal print modes.

The numbers, n1 & n2, combine together to determine the value.
n2 is equal to the number of times 256 will divide into the desired
value, and n1 is the remainder. For example, if you wanted to pr
500 dots per row, 500/256=1 with 244 remaining, so the set-up st
to print 500 dots per row in double density is:

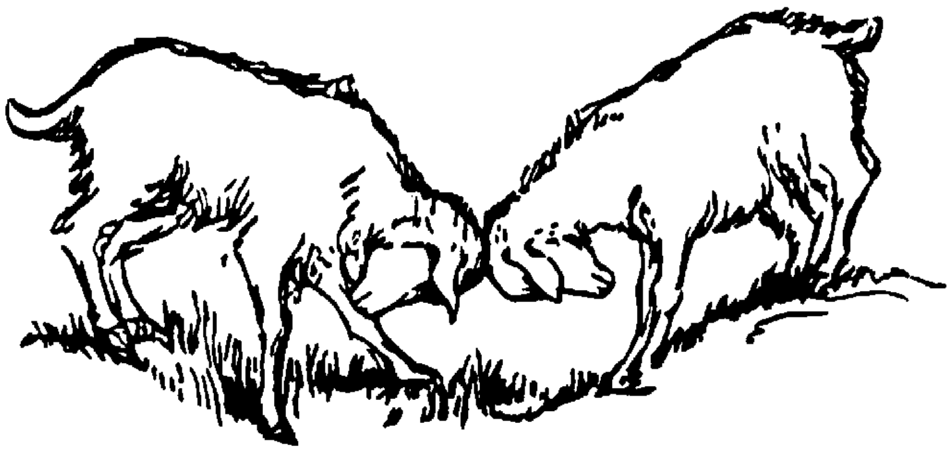
CHR$(27)&"L"&CHR$(244)&CHR$(1)

In my program example I print 60 dots in single density. Si
60/256<0, n2=0 and n1 is the total number of dots. This is true
any value under 256.

If the number of dot columns will change in your application,
the best thing is to initialize two variables, to calculate n1 and n2
for any width, as shown:

```
10 X=Z-(INT(Z/256)*256)
20 Y=INT(Z/256)
30 SET$=CHR$(27)&"L"&CHR$(X)&CHR$(Y)
...
...
...
100 Z=500
110 PRINT #1:SET$
...
...
...
200 Z=140
210 PRINT #1:SET$
```

This may sound like a crazy way to send a number to the printer,
but it's necessary because the data must be sent as Hexidecimal
numbers, two digits per number. The largest two-digit Hex number is
>FF, which equals 255, so everything goes in chunks of 255 or less.



LOAD INTERRUPTS

The Rest of the Meal-Appendixes

TI Product Sources
compiled by Ron Albright

What follows is a sort of "who's who" and "where to look" for various support services for the TI 99/4A Home Computer. The list will include commercial concerns, and public (often free) sources of information. It is certainly not comprehensive but, the author has made every effort to check information such as addresses and phone numbers, but cautions the reader that these are subject to change.

Hardware

Corcomp, Incorporated
2211-G Winston Road
Anaheim, California 92806
(714) 956-4450

AEI
P.O. Box 10306
Rockville, Maryland 20850

Myarc, Inc.
241 Madisonville Road
Basking Ridge, New Jersey 07920

Captain's Wheel
17295 Chippendale
Farmington, MN 5024
(612) 460-6348

Miller's Graphics
1475 W. Cypress Avenue
San Dimas, California 91773
(714) 599-1431

Rave 99
23 Florence Road
Bloomfield, CT 06002

Ryte Data
210 Mountain Street
Haliburton, Ontario
Canada K0M 1S0
(705) 457-2774

Dijit Systems
4345 Hortensia Street
San Diego, California 92103
(619) 295-3301

Horizon Computer Limited
P.O. Box 554
Walbridge. Ohio 43465

Software

Tigercub Software
156 Collingwood Avenue
Columbus, Ohio 43213
(614) 235-3545

Ryte Data
210 Mountain Street
Haliburton, Ontario, Canada
(705) 457-2774

Asgard Software
P.O. Box 10306
Rockville, Maryland 20850

Quality 99 Software
1884 Columbia Road #500
Washington, DC 20009

Millers Graphics
1475 West Cypress Avenue
San Dimas, California 91773

DataBioTics
P.O. Box 1194
Palos Verdes Estates, Calif. 90274

Inscebot, Inc.
P.O. Box 260
Arnold, Maryland 21012

CSI Design Group
P.O. Box 50150
St. Louis, Missouri 63105

DataBioTics
P.O. Box 1194
Palos Verdes Estates, CA. 90274

SST Software
Box 26
Cedarburg, Wisconsin 53012
(414) 771-8415

McCann Software
P.O. Box 34160
Omaha, Nebraska 68134

Heim Industries
P.O. Box 296
Clifton Park, NY 12065

Bright Micro Komputers
2781 Resor Road
Fairfield, Ohio 45014

Trinity Systems
1022 Grandview
Pittsburg, PA 15237

Great Lakes Software
804 E. Grand River Avenue
Howell, Michigan 48843

Intelpro
5825 Baillargeon Street
Brossard, Quebec, Canada J4Z 1T1
(514) 656-8798

Mail-Order Distributors

Bits and Chips
23637 HWY 99
Edmonds, Washington
(206) 775-7390

Tenex Computer Express
P.O. Box 6578
South Bend, Indiana 46660
(219) 259-7051

Triton Products, Inc.
P.O. Box 8123
San Francisco, Calif. 94128
1-800-227-6900

Computer Micro Products
2460 Wisconsin Avenue
Downers Grove, Illinois 60515
(312) 960-1950

Ramsoft Enterprises
1501 East Chapman Avenue
Suite 338
Fullerton, California 92631

Hunter Electronics
604 S. Fairview Avenue
Elmhurst, Ill. 60126
(312) 832-6558

Specialist In
821 Excelsior
Hopkins, Minnesota
(612) 938-3161

Tex-Comp
P.O. Box 33034
Granda Hills, Ca 91344
(818) 366-6631

Pilgrim's Pride
219 N. York Rd.
Hatboro, PA 19040
(215) 441-4262

Texaments
53 Center Street
Patchogue, N.Y. 11772

Replacement Parts/Surplus:

Arnold Company
P.O. Box 512
Commerce, TX 75428
(214) 395-2922
(Keyboards)

Lolir
13933 N. Central #212
Dallas, TX 75243
(214) 234-8056
(Keyboards, Power Supplies, cassette cables)

Repairs

Daymon Fikes
Texas Instruments Attn:Repair
2305 N. University
Lubbock, TX 79415
(806) 741-2321

National Organizations (Membership fees required)

99/4A National Assistance Group
Box 290812
Fort Lauderdale, Florida 33329
(305) 583-0467

99 Users Group Association
3535 South H. Street
Bakersfield, Calif. 93304
(805) 397-4361

National Non-Profit Assistance (Information, Public-Domain Software)

Amnion Helpline
116 Carl Street
San Francisco, Calif. 94117
(415) 753-5581

National Telecomunication Networks

Compuserve
5000 Arlington Center Blvd.
P.O. Box 20212
Columbus, Ohio 43220
(800) 848-8990

The Source
1616 Anderson Road
McLean, Virginia 22102
(800) 336-3330

GEnie
401 N. Washington Street
Rockville, Maryland 20850
1-800-638-9636

Delphi
3 Blackstone Street
Cambridge, Mass 02139
1-800-544-4005

Bulletin Board Software

Commercial

BBS System
1411 N. 36th
Melrose Park, Il. 60160

Freeware:

TIBBS(tm)
P.O. Box 383
Kennesaw, Georgia 30144
(404) 425-5254

Techie Bulletin Board
Monty Schmidt
525 Wingra Street
Madison, Wisconsin 53714

Scott Darling
W. 5515 Woodside
Spokane, Washington 99208

Public Domain

TI-COMM

John Clulow
345 West South Boundary
Perrysburg, Ohio 43551

Pro-99er BBS
Mark Hoogendoorne
21 Long Street
Burlington, Massachusetts 01803

Publications

Micropendium
P.O. Box 1343
Round Rock, Texas 78680
(512) 255-1512

Smart Programmer
171 Mustang Street
Sulphur, Louisiana 70663

Computer Shopper
407 S. Washington Avenue
P.O. Box F
Titusville, Florida 32781
(305) 269-3211

TRAVelER Genial Computerware
835 Green Valley Drive
Philadelphia, Pennsylvania 19128
($30/year, 6 issues of a "magazine on disk"; a flippy disk with 720 sectors of programs/utilities)

R/D Newsletter
210 Mountain Street
Haliburton, Ontario
Canada K0M 1S0

Fairware Listing

[Note: It is strongly suggested by the author to contact by mail the authors listed below inquiring whether their offerings are still available before sending disk or payment. Further, include a self-addressed, stamped envelope to ensure a timely reply. While the author has made every effort to provide factual and current information in this listing, he cannot assume any liability for problems encountered as a result of inadvertent errors.]

MASSCOPY Steve Lawless 2514 Maple Avenue, Wilmington, Delaware 19808
Disk cloner

NEATLIST Danny Michaels Route 9, Box 460 Florence, AL. 35630
Programming utility

SCREENDUMP Danny Michaels (See Above)
Screen dump printer utility

FAST-TERM Paul Charlton 1110 Pinehurst Court Chalottesville, VA 22901
Terminal Emulator

EASYSPRITE Tom Freeman 515 Alma Real Dr.,Pacific Palisades, CA 90272
Programming utility for graphics

DISASSEMBLER Marty Kroll 218 Kaplan Avenue Pittsburg, P. 15227
Disassembler programming utility

Checkbook and Budget Manager by John Taylor, 2170 Estaline Drive, Florence, Alabama 35630;Household budgeting and finance management

Games; John Taylor (see above address); collection of commercial quality games and educational activities for children and adults. $5 or disk and mailer and return postage.

Director '99' Robert Neal/Ed Bert, P.O. Box 216R, Romeoville, Il. 60441
Disk catalog database program

PRBase William Warren, 2373 Ironton Street, Aurora, Colorado 80010
Database program

Best Songs 1 and 2; Music written by Bill Knecht, 815 Yorkshire, Pasadena, Texas 77503. $5 for one disk, $8 for both.

Doors to Eden and First Days in Eden; Steven Cheairs, P.O. Box 27547, Albuquerque, NM 87125. Two games to be used with the TI Adventure Module. Author asks a $2 donation at the time the games are ordered; 2 disks or 2 cassettes.

Sprint Utility; Ken Houle, 27721 W. Wakefield Rd., Saugus, CA 91350. Utility for Aseembly programmers to dump DV80 files to printer or disk.

Printout; Steven Mehr, 633 Hollyburne Lane, Thousand Oaks, CA 91360. XB utility to printout DV80 files wit printers with many options. $5 plus disk and mailer.

Pilot 99; by Tom Weithofer (deceased). As a service to the TI community, Dr. Jim McCulloch (9505 Drake Avenue, Evanston, IL 60203) will send a copy of this fine programming language to anyone who sends two disks, self-addressed mailer and sufficient postage.

Fas-Tran; Bill Harms, 6527 Hayes Court, Chino, CA 91710; checkboo/spreadsheet program with easy linking to Multiplan.

Weather Forecaster; Garry Cox, 3174 Melbourne, Memphis, TN 38127; Graphics and home weather prediction.

For a complete list of Freeware, send $1 to MICROpendium, P.O. Box 1343, Round Rock, Texas 78680. The list they have is verified and up-to-date and now is 10 pages long. Support this valuable source of TI software - support Freeware.

User Group List
Courtesy of Art Byers and the
Westchester 99er's User Group

Summit 99ers Users Group P. O. Box 3201 Cuyohoga Falls, Ohio 44223	216-456-0450
Fox City Users Group P. O. Box 2553 Appleton, Wisconsin 54913	414/766/3515
Atlanta 99/4A Computer Users P. O. Box 19841 Atlanta, Georgia 30325	404-233-3096
North New Jersey TI Users Group 16 Judith Ann Drive Ringwood, New Jersey 07456	
Mid Illinois Computer Resource P. O. Box 766 Bloomington, Illinois 61701-0766	309/962/9305
Boise 99'ers Computer Club 1331 Colorado Avenue Boise, Idaho 83706	208/344/1409
NorthEast Iowa Home Computer User' 1528 Longfellow % Terry Maxfield Waterloo, Iowa 50703	
Central Ohio Ninety-Niners, Inc. 8055 Simfield Road % D. Heim Dublin, Ohio 43017	614/868/0632
Airport Area Computer Club P. O. Box 710 Coraopolis, Penn. 15108	
The Daytona 99ers P. O. Box 15232 Daytona Beach, Florida 32015	904/427/8532
Rocky Mountain 99ers P. O. Box 12605 Denver, Colorado 80212	303/759/0699
Central Iowa 99/4A Users Group P. O. Box 3043 Des Moines, Iowa 50316	515/266/7788

Users Group of Orange County 213/439/0785
17301 Santa Isabel Street
Fountain Valley, Calif.92708

L A 99ers Computer Group 213/439/0785
P. O. Box 3547
Gardenia, California 90247-7247

99/4A Minn & Dakota Home User 701/772/6180
509 Reeves Drive
Grand Forks, N. Dakota 58201

Grand Rapids 99er Users Group 616/791/0059
1419 Laughlin Dr. N. W.
Grand Rapids, Michigan 49504-2423

Johnson Space Center Users Group ???/337/4110
2321 Coryell St. % John Owen
League City, Texas 77573

Net 99er Home Computer Group 817/656/1473
P. O. Box 534
Hurst, Texas 76053

Kankakee TI Users Group
P. O. Box 1945
Kankakee, Illinois 60901

Mid Atlantic Ninety Nine'ers 703-777-2017
P. O. Box 267
Leesburg, Virginia 22075

Titex TI Users Group 516/796/8359
36 Fox Place
Hicksville, New York 11801

Kentuckiana 99/4A Computer Society 812/923/3888
P. O. Box 36246
Louisville, Kentucky 40233-6246

St. Louis 99ers 314/428/0752
P. O. Box 63158
St. Louis, Missouri 63163

Greater Orlando 99ers Users Group 305-293-0769
P. O. Box 1381
Maitland, Florida 32751

Greater Omaha TI-99/4A User's Grou 402/556/0702
11215 Crippen Circle
Omaha, Nebraska 68138

Pomona Valley 99/4A Users'Group 714/984-4107
1833 E. Princeton St. % C. Perez
Ontario, California 91764

Miami County Area 99/4A Users Grou 219/563/2213
P. O. Box 1194
Peru, Indiana 46970

Bluegrass 99/4 Computer Society 606/268/0210
P. O. Box 11866
Lexington, Kentucky 40578-1866

T. I. Users Group of Will County 815/886/6552
P. O. Box 216R
Romeoville, Illinois 60441

Great Lakes Computer Group 313/623-7926
P. O. Box 7151
Roseville, Michigan 48305

MSP 99 Users Group 612/429-5256
P. O. Box 12351
St. Paul, Minnesota 55112

The Suncoast Beeper 813/347-6942
8421 Westridge Drive
Tampa, Florida 33615

N. W. Suburban 99 Users Group 312/980/9234
[illegible]1 Freeman Road
Hoffman Estates, Ill. 60195

Upper Pinellas 99'er User Group 813/736-1616
P. O. Box 3031
Seminole, Florida 33542

Mid/America 99 Users Group
5936 Hardy
Merriam, Kansas 66202

Lima Area 99/4A User Group
2225 High Ridge
Lima, Ohio 45805

New Jersey Users Group 201/686/5619
49 Pine Grove Ave. % Mel Gary
Somerset New Jersey 08873

W.W. 99'ers of Champaign/Urbana 217-344-5281
2020 Rebecca Drive
Champaign, Illinois 61821

Tidewater 99/4 Users Group Inc. 804/596-6450
P. O. Box 1935
Newport News, Vir. 23601

Milwaukee Area 99-4 User Group 414/264/4735
4122 North Glenway
Wauwatosa, Wisconsin 53222

Cin-Day User's Group 513/777/0110
P. O. Box 519
West Chester, Ohio 45069-0519

K-Town 99/4A Users Group
116 Richards Drive
Oliver Springs, Tenn 37840

Shoals 99'ers
P. O. Box 2928 205-776-2032
Muscle Shoals, Alabama 35662

Long Island 99'er Users Group 516-587-5462
P. O. Box 544
Deer Park, New York 11729

West Penn 99ers 412-271-6283
R. R. 1 Box 73A % J. F. Willforth
Jeanette, Pennyslvania 15644

Wiregrass 99'er User's Group
Att.Newsletter 102 Auburn Dr.
Enterprise, Alabama 36330

Jacksonville TI-99/4a Users Group 501/982/9710
P. O. Box 525
Jacksonville, Arkansas 72076

Bayou 99 Users Group 318/477/3687
P. O. Box 921
Lake Charles, Louisiana 70602

Brazos Valley 99'ers
P. O. Box 7053 817/848/4589
Waco, Texas 76714

Edmonton Users Group 403/467/6021
P. O. Box 11983
Edmonton, Alberta Cana T5J 3L1

The Ottawa T. I. 99/4 Users Group 613/837-1719
P. O. Box 2144 Station D
Ottawa, Ontario Canada K1P 5W3

Wichita 99er's Users Group 316/221/7148
R.R.5 Box 13 % Guy Hulsey
Winfield, Kansas 67156

Lower Michigan 99/4A users Group
P. O. Box 885
Troy, Michigan 48099

NorthWest Ohio 99'er News 419/666/4945
5926 Ranbo Lane
Toledo, Ohio 43623

The Central Westchester 99'ers Clu 914/961-5993
1261 Williams Dr. % A. Byers
Scrub Oak, New York 10588

Magnetic (TI 99/4A)
57 River Road
Andover, Massachusetts 01810

Amarillo 99/4A User Group 806/359-0380
P. O. Box 8421
Amarillo, Texas 79114-8421

Quad Cities Computer Club
P. O. Box 1124
Bettendorf, Iowa 52722

Boston Computer Society TI user Group 617/353/7369
One Center Plaza % J. P. Hoddie
Boston, Massachusetts 02108

The Windy City 99 Club 312/337/5997
640 N. LaSalle R. 280 % M. Mickels
Chicago, Illinois 60610

Corpus Christi 99ers Users Group 512-852-4874
5205 Tartan
Corpus Christi, Texas 78403

Decatur 99er Home Computer Users 217/877-1631
P. O. Box 726
Decatur, Illinois 62525

San Diego Computer Society T I Sig 619-296-9386
P. O. Box 83821
San Diego, California 92138

Southern California Computer Group 619/462/5802
P. O. Box 21181
El Cajon, California 92021

Eugene 99/4A User Group 503-747-1768
P. O. Box 11313
Eugene, Oregon 97440

The Forest Lane Users Group 4413 Cornell Drive % J. Gillo Garland, Texas 75042	214-480-1302
Mid-South 99/4A Users Group P. O. Box 38522 Germantown, Tennessee 38183-0522	901/363/6273
Nutmeg TI-99ers 10 Jolly Road % J. Ryan Ellington, Connecticut 06029	203/875/1647
Hoosier Users Group P. O. Box 2222 Indianapolis, Indiana 46206-2222	317/631/7255
Dallas TI Home Computer Group 1221 Mosswood Place Irving, Texas 75061	214/239-6829
Kansas City TI99/4A Computer U. G. P. O. Box 12591 North Kansas City, Mo. 64116	913-371-1092
Syracuse T I 99/4A Users Group 144 Hillside Way Camillus, New York 13031	315/625/4409
Lincoln 99 Computer Club 4501 South 50th Street Lincoln, Nebraska 68516	402-489-2364
Puget Sound 99ers P. O. Box 6073 Lynnwood, Washington 98032	
Madarea 99'ers 437 W. Gorham % Wisc. Blue Print Madison, Wisconsin 53703	608/648-2883
Miami 99/4A Users Group 19301 NE 19th Avenue N. Miami Beach, Florida 33179	305-257-2102
Texas Instruments Baltimore Users Group P. O. Box 3 Perry Hall, Maryland 21128	
Fox Valley Users Group 34W.762 S. James Drive St. Charles, Illinois 60174	312/931/0360

Siouxland 99'ers 4604 Bluestem Circle Sioux Falls, South Dak 57106	605/338/7050
The Michiana 99/4A User's Group 911 Dover Drive South Bend, Indiana 46614	219/277/1990
Ninety-Niners of the Vancouver Are P. O. Box 508 Vancouver, Washington 98666	206/693/7070
Northern NJ Ninty Niner Users Group P. O. Box 515 Bedminister,, New Jer. 07921515	201-234-1488
The Downeast 99er's P. O. Box 542 Westbrook, Maine 04092	
Cleveland Area 99/4A Users Group P. O. Box 23283 Euclid, Ohio 44123	216-274-2544
Cross Roads 99ers Computer Group P. O. Box 293 York, Nebraska 68467	
Pekin Users Group 559 Chicago Street East Peoria, Illinois 61611	
Carnation City 99'ers User Group 205 Fernwood Blvd % D. S. Brain Alliance, Ohio 44601	216/823/8958
Greater Dayton 99'ers P. O. Box 248 Englewood, Ohio 45322-0248	513/836/5918
San Fernando 99ers P. O. Box 1844 Canyon Country, Calif 91351	818-507-6219
The Fort's User Group P. O. Box 11212 Fort Wayne, Indiania 46856-1212	219-432-1228

Peeks and POKES

Compiled by Scott Darling
GEnie ID TIKSOFT

24K OF DATA STORAGE

If you need to work with quite a bit of data or would like to change programs, but save the data after you press CALL QUIT then you can set up the 24K of High-Memory in the PEB as a single data file called "EXPMEM2", you open this file just as you would a disk file with one exception - you must PRECEED th OPEN statement with a CALL LOAD to the location -24574 as follows:

For INT/VAR files - 24
For DIS/VAR files - 16
For INIT/FIX files - 8
For DIS/FIX files - 0

Heres and example:
If you want to open up the Expansion Memory for Display,Variable 80 files this is what you'd do:

```
100 CALL INIT
110 CALL LOAD(-24574,16)
120 OPEN #1:"EXPMEM2",RELATIVE,UPDATE,DISPLAY,VARIABLE 80
```

Then continue on as you normally would.

If you want to store both data and assembly language routines at the same time do this:

```
100 CALL INIT
110 CALL LOAD(-24574,-16)
120 OPEN #1:"EXPMEM2"
130 CALL LOAD ("DSK1.ASSM1")
140 CALL LOAD ("DSK2.ASSM2")
150 CALL LINK ("START")
160 REM CONTINUE REST OF PROGRAM
```

In the above example the 24 K of high-memory was saved for use as a DATA file (DIS/VAR 80 format) then the assembly routines were loaded. The computer wll look for the best place to put the routines and will adjust the pointer accordingly. After the routines are loaded, a LINK statement starts the first rutine and off we go.

If that's not enough for you, you can also use the MINI-MEMORY for 4K more of storage of assembly routines! Now that's 16K of program space, 12K of assembly routine space!
****

These are all of the Peeks & Pokes that I have come across for use with X-Basic and 32K memory expansion (be sure to do a "CALL INIT"). The P & Q variables are used for "PEEK" - the numbers are for "POKE" or "LOAD".
****

ADDRESS	, VALUE(S)	MEANING IN EXTENDED BASIC
		CALL VERSION(X) IF X=100 100= NEWEST VERSION OF X/B CART
8192	, P	USE (PEEK,P) IF P<> 70 OR <>121 THEN DO A CALL INIT
8194	,	FIRST FREE ADDRESS IN LOW MEMORY
8196	,	LAST FREE ADDRESS IN LOW MEMORY
-28672	, P	P=0 SPEECH NOT ATTACHED P=96 OR P=255 SPEECH IS ATTACHED
-31572	, 0 TO 255	VARY KEYBOARD RESPONSE
-31740	, P , Q	PUT IN DIFFERENT TO CHANGE BEEPS,WARNINGS, ETC
-31744	, 0 TO 15	CONTINUATION OF LAST SOUND (0=LOUD AND 15=SOFT)
-31748	, 0 TO 255	CHANGE THE CURSOR FLASHING AND RESPONSE TONE RATES
-31788	, 160	BLANK OUT THE SCREEN (MUST PUSH A KEY TO ACTIVATE)
	, 192	NO AUTOMATIC SPRITE MOTION OR SOUND
	, 224	NORMAL OPERATION
	, 225	MAGNIFIED SPRITES
	, 226	DOUBLE SIZE SPRITES
	, 227	MAGNIFIED & DOUBLE SIZED SPRITES
	, 232	MULTICOLOR MODE (48 BY 64 SQUARES)
-31794	, P	TIMER FOR CALL SOUND (COUNTS FROM 255 TO 0)
-31804	, X , Y	RETURN TO THE TITLE SCREEN (USE "PEEK (2,X,Y)")
	, P	CHANGE THE CURSOR FLASH RATE (0 TO 255)
-31806	, 0	NORMAL OPERATION
	, 16	DISABLE QUIT KEY (FCTN =)
	, 32	DISABLE SOUND (USE NEG DUR FOR CONTINOUS SOUND)
	, 48	DISABLE SOUND & QUIT KEY
	, 64	DISABLE AUTO SPRITE MOTION
	, 80	DISABLE SPRITES & QUIT KEY
	, 96	DISABLE SPRITES AND SOUND
	, 128	DISABLE ALL THREE
-31808	, P , Q	DOUBLE RANDOM NUMBERS (0 TO 255) NEED "RANDOMIZE"
-31860	, 4	GO FROM EX-BASIC TO CONSOLE BASIC (NEED "NEW")
	, 8	AUTO RUN OF DSK1.LOAD
-31866	, P , Q	END OF CPU PROGRAM ADDRESS (P*256+Q)
-31868	, 0	NO "RUN" OR "LIST" AFTER "BREAK" IS USED
	, 0 , 0	TURNS OFF THE 32K MEMORY EXPANSION
	, 255 , 231	TURNS ON THE 32K MEMORY EXPANSION
-31873	, 3 TO 30	SCREEN COLUMN TO START AT WITH A "PRINT"
-31877	, P	P&32 = SPRITE COINCIDENCE P&64 = 5 SPRITES ON A LINE
-31878	, P	HIGHEST NUMBER SPRITE IN MOTION (0 STOPS ALL)
-31879	, P	TIMER FOR VDP INTERRUPTS EVERY 1/60 OF A SEC (0 TOP 255)
-31880	, P	RANDOM NUMBER (0 TO 99) NEED "RANDOMIZE"
-31884	, 0 TO 5	CHANGE KEYBOARD MODE (LIKE "CALL KEY(K,...)")
-31888	, 63 , 255	DISABLE ALL DISK DRIVES (USE "NEW" TO FREE MEMORY)
	, 55 , 215	ENABLE ALL DISK DRIVES (USE "NEW" TO FREE DRIVES)
-31931	, 0	UNPROTECT X-B PROTECTION
	, 2	SET "ON WARNING NEXT" COMMAND
	, 4	SET "ON WARNING STOP" COMMAND
	, 14	SET "UNTRACE" COMMAND
	, 15	SET "UNTRACE" COMMAND & "NUM" COMMAND
	, 16	SET "TRACE" COMMAND
	, 64	SET "ON BREAK NEXT" COMMAND
	, 128	PROTECT X/B PROGRAM

-31952	P	PEEK P=55 THEN 32K EXPANSION MEMORY IS OFF <>55 MEANS ON
-31962	32	RETURN TO THE TITLE SCREEN
	255	RESTART X/B W/DSK1.LOAD
-31974	P , Q	END OF VDP STACK ADDRESS (P*256+Q)
-32112	8	SEARCHES DISK FOR ?
-32114	2	RANDOM GARBAGE
	13	SCREEN GOES WILD
	119	PRODUCE LINES
-32116	2	RANDOM CHARACTERS ON SCREEN
	4	GO FROM X/BASIC TO BASIC
-32187	0	UNPROTECT XB PROGRAM
	2	SET "ON WARNING NEXT" COMMAND
	4	SET "ON WARNING STOP" COMMAND
	9	SET 0 LINE NUMBER
	14	SET "UNTRACE" COMMAND
	15	SET "UNTRACE" COMMAND & "NUM" COMMAND
	16	SET "TRACE" COMMAND
	64	SET "ON BREAK NEXT" COMMAND
	128	PROTECT XB PROGRAM
-32188	1	CHANGE COLOR AND RECEIVE SYNTAX ERROR
	127	CHANGE COLOR AND RECEIVE BREAKPOINT
-32630	128	RESET TO TITLE SCREEN
-32699	0	UNPROTECT XB PROGRAM
	2	SET "ON WARNING NEXT" COMMAND
	4	SET "ON WARNING STOP" COMMAND
	14	SET "UNTRACE" COMMAND
	15	SET "UNTRACE" & "NUM" COMMAND
	16	SET "TRACE" COMMAND
	64	SET "ON BREAK NEXT"
	128	PROTECT XB PROGRAM
-32700	0	CLEARS CREEN FOR AN INSTANT
-32729	0	RUN "DSK1.LOAD"
-32730	32	RESET TO TITLE SCREEN
-32961	51	RESET TO TITLE SCREEN
	149	SETS "ON BREAK GOTO" LOCKS SYSTEM

The follwoing Loads require E/A or Minimemory:

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

* PASCAL LOADS

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

TI Console Memory Map
Compiled by Robert Coffee

```
+-------------------------------+
|Communications Register Unit 8K|
+-------------------------------+
```

Let's run down the CRU again.

>0000-03FE	CRU TMS 9901 space, required.
>0404-10FE	For test equipment use on production line.
>1100-11FE	Disk Controller.
>1200-12FE	Modem.
>1300-13FE	Primary RS232, serial ports 1 & 2 and parellel port #1.
>1400-14FE	Unassigned.
>1500-15FE	Secondary RS232, serial ports 3 & 4 and parellel port #2.
>1600-16FE	Unassigned
>1700-17FE	Hex-bus (tm).
>1800-18FE	Thermal printer.
>1900-19FE	EPROM programmer, something that TI planned but never came out with.
>1A00-1AFE	Unassigned
>1B00-1BFE	Unassigned
>1C00-1CFE	Video Controller Card.
>1D00-1DFE	IEEE 448 Controller Card,apparently something else that TI didn't release.
>1E00-1EFE	Unassigned
>1F00-1FFE	P-Code Card.

```
+------------+
| VDP RAM 16K |
+------------+
```

>0000-02FF	SCREEN IMAGE TABLE (.75K) This portion of VDP Ram contains the characters that you see on your screen. Hex 0000 is the character in the top-left corner of the screen. The ascII values have offset value of >60.
>0300-036F	SPRITE ATTRIBUTE TABLE (.1K) This table holds the information for all 28 sprites. eg. position(dot row, dot column), character number, and its color.
>0370-077F	PATTERN DESCRIPTOR & SPRITE PATTERN TABLE (1K) Contains the patterns for characters & sprites. eg. address for the space is (768+8*32=1024).
>0780-07FF	SPRITE MOTION TABLE (.12K) This holds row and column velocities for all 28 sprites and it used by the Interrupt routine in console ROM. The routine executes 60 times a second(or 60 Hertz)and since it is interrupt driven it will use the values i this table to update the Sprite Attribute Table. Each sprite uses 4 bytes. One for row velocity, one for column velocity, and 2 for the system to use.

```
>0800-081F  COLOR TABLE (.03K)
            This portion contains the foreground and backround color
            information for each character set. The definition for each color
            uses one byte, bytes 0-3 for foreground and 4-7 for backround.
            There are 32 bytes in the table.(Sets 1-32). Sets 1-3 aren't used
            Set 4(in table) is character set 0, set 5 is 1, etc. up to set
            18(for table) 14 for character set. Sets 19-32 aren't used by the
            'COLOR' statement in Extended BASIC.
>0820-35D7  DYNAMIC MEMORY SPACE (11.5K)
            This holds your program and other things like PAB(Peripheral Access
            Block), strings, symbol table, numeric value table,& the line
            number table(for finding the lines of your program thats in the
            crunched format).Your BASIC program is loaded from >35D7(bottom)
            and up.Lines appear as they as typed in, not in the order of line
            numbers (like 100,110,120).
>35D8-3FFF  FILE BUFFERS (2.5K)
            CALL FILES(n) will change this starting address but with CALL FILES
            (3) it start repectively at >35D8. If the power up routine finds a
            disk controller then the computer will automatically reserve this
            this space for drive control, file allocation, and data buffering.
```

```
+-----------------+
| Console GROM 18K |
+-----------------+
```

There are 3 GROM chips in our consoles. Each has 8K of space but only 6K is used. The difference between ROM and GROM is that GROM automaticallt increments itself everytime it is accessed.GROM is also written in GPL (Graphics Programming Language),which TI wrote themselves. Here are those 3 GROM chips:

```
GROM 0 >0000-17FF The title screen power up routine, title screen character
                  set, standard character set(Upper & Lower casd), cassette DSR
                  messages and the trigonometric functions.

GROM 1 >2000-37FF Vector tables for BASIC, the error messages, and part of the
                  BAISC interpreter.

GROM 2 >4000-57FF Part of the BASIC interpreter,the reserved word list and
                  their associated token values.
```

GROM chips 3-6 (24K)are in the Extended BAISC cartridge and contain the following:

```
GROM 3 >6000-77FF X/BASIC vector tables, the error statements for X/BASIC and
                  part of the X/BASIC interpreter.

GROM 4 >8000-97FF Part of the X/BASIC interpreter.

GROM 5 >A000-B7FF Part of the X/BASIC interpreter.

GROM 6 >C000-D7FF Part of the X/BASIC interpreter, the reserved word list and
                  their associated token values.
```

```
        +--------------------------------------------------+
        |   Video Display Processor RAM for Extended BASIC |
        |                                                  |
        |             VDP , a complete look.               |
        +--------------------------------------------------+

+------------------------------------------------------------------+
|>0000          VDP SCREEN IMAGE TABLE                   768 bytes |
|                                                                  |
|    each screen location takes up 1 byte, the character          |
|    value at each location is offset by >60.                     |
|                                                                  |
|    LOCATION=COL+32*(ROW-1)                                       |
|                                                      D           
+------------------------------------------------------------------+
|>0300          SPRITE ATTRIBUTE TABLE                   112 bytes |
|                                                                  |
|    Each sprite takes up 4 bytes. (room enough for only 28)      |
|    These for bytes consist of vertical postion -1, horizontal   |
|    position, character # + >60, clock bit, color.               |
|                                                            >036F |
+------------------------------------------------------------------+
|>0370          EXTENDED BASIC SYSTEM BLOCK                        |
|                                                                  |
|      >0371 Auto Boot (needed flag)                               |
|      >0372 Line to start execution at                            |
|      >0376 Saved symbol table "GLOBAL" pointer (used with        |
|            subprograms).                                         |
|      >0378 Used for CHR$                                         |
|      >0379 Sound blocks                                          |
|      >0382 Saved program pointer for continue and text pointer   |
|            for break                                             |
|      >0384 Saved buffer level for continue                       |
|      >0386 Saved expansion memory for continue                   |
|      >0388 Saved value stack pointer for continue                |
|      >038A ON ERROR line pointer                                 |
|      >038C Edit recall start address                             |
|      >038E Edit recall end address                               |
|      >0390 Used as temporary storage place                       |
|      >0392 Saved main symbol table pointer                       |
|      >0394 Auto load temp for inside error                       |
|      >0396 Saved last subprogram pointer for continue            |
|      >0398 Saved ON WARNING/BREAK bits for continue              |
|      >039A Temp to save subprogram table                         |
|      >039C Same as above but used in subprograms                 |
|      >039E Merged temp for PAB (Peripheral Access Block)pointer  |
|      >03A0 Random number generator seed 2                        |
|      >03A5 Random number generator seed 1                        |
|      >03AA Input temp for pointer to prompt                      |
|      >03AC Accept temp pointer                                   |
|      >03AE Try again(used when you input a string instead of a   |
|            number)                                               |
|       >03B0 Pointer to standard string in VALIDATE                      |
|       >03B2 Length of standard string in VALIDATE                       |
|       >03B6 Size temp for record length. Also temp in relocating        |
|             program                                                     |
|       >03B7 Accept "TRY AGAIN" flag                                     |
|       >03B8 Saved pointer in SIZE when "TRY AGAIN"                      |
|       >03BA Used as temp storage place                                  |
|       >03BC Old top of memory for relocating program / temp for         |
|             INPUT                                                       |
|       >03BE New top of memory for relocating program                    |
|       >03C0 Roll out area for scratch pad RAM when certain              |
|             operations are performed                                    |
|       >03DC Floating point sign                                         |
|                                                                   >03EF |
+-------------------------------------------------------------------------+
|>03F0          PATTERN DESCRIPTOR TABLE                        912 bytes |
|             / SPRITE DESCRIPTOR TABLE                                   |
|                                                                         |
|     Each character take up 8 bytes. There are 114 characters            |
|     here. They are numbered from 30 to 143.                             |
|                                                                   >077F |
+-------------------------------------------------------------------------+
|>0780          SPRITE MOTION TABLE                             128 bytes |
|                                                                         |
|     Each sprite takes up 4 bytes. These 4 bytes contain the             |
|     vertical velocity, horizontal velocity, & the last 2 are            |
|     for system use.                                                     |
|                                                                   >07FF |
+-------------------------------------------------------------------------+
|>0800          COLOR TABLE                                      32 bytes |
|                                                                         |
|     Each character set requires only 1 byte. This byte is              |
|     broken up into the foreground & backround.                          |
|                                                                   >081F |
+-------------------------------------------------------------------------+
|>0820          CRUNCH BUFFER                                   160 bytes |
|                                                                         |
|     This area of VDP is used when the system needs to crunch            |
|     ASCII values into token codes.                                      |
|                                                                   >08BE |
+-------------------------------------------------------------------------+
|>08C0          EDIT / RECALL BUFFER                            152 bytes |
|                                                                         |
|     What you type in at the command line is stored here.                |
|                                                                   >0957 |
+-------------------------------------------------------------------------+
|>0958          VALUE STACK                                      16 bytes |
|                                                                         |
|     Used by these ROM routines : SADD, SSUB, SMUL, SDIV, & SCOMP        |
|                                                                   >0967 |
+------------------------------------------------------------+
|>0968                                            11888 bytes |
|                                                             |
|    The items in this area move according to the size of the |
|    crunched program & the system always reserves 48 bytes of|
|    area.                                                    |
|                                                             |
|    The SYMBOL TABLES are generated during the pre-scan peroid
|    after you type RUN. The strings are placed into memory when
|    they are assigned.                                       |
|                                                             |
|    WITHOUT MEMORY EXPANSION:                                |
|                              -STRINGS                       |
|                              -DYNAMIC SYMBOL TABLE & PABS   |
|                              -STATIC SYMBOL TABLE           |
|                              -LINE NUMBER TABLE             |
|                              -PROGRAM SPACE(crunched program)
|                                                             |
|   WITH MEMORY EXPANSION:                                    |
|                           -STRINGS                          |
|                           -DYNAMIC SYMBOL TABLE & PABS      |
|                           -STATIC SYMBOL TABLE              |
|                           -Numeric values, line number table,
|                            & program space are moved into   |
|                            High-memory expansion( >A000 )   |
|                                                       >37D7 |
+------------------------------------------------------------+
|>37D8         DISK BUFFER AREA [ default 'CALL FILES(3)' ]  5 bytes
|                                                             |
|       >37D8 Validation code for the disk controller DSR ( >AA )
|       >37D9 Points to TOP of VDP memory ( >3FFF )           |
|       >37DB CRU base identification                         |
|       >37DC Maximum number of OPENed files ( >03 default )  |
|   -------------------------------------------------------   |
|           File Control Block for 1st file OPENed   518 bytes|
|                                                             |
|       >37DD Current Logical record offset                   |
|       >37DF Sector number location of File Descriptor Record|
|       >37E1 Logical Record Offset(used woth VARIABLE files only)
|       >37E2 Drive number(using the high order bit)          |
|    File Descriptor Record(brought from the disk 256 bytes)  |
|       >37E3 File name                                       |
|       >37ED Reserved ( >0000 )                              |
|       >37EF File status flags(file type & write protection) |
|       >37F0 Max number of records per Allocation Unit(1 AU=1 Sector)
|       >37F1 number of sectors currently allocated (256 byte blocks)
|       >37F3 End of File offset within the last used sector  |
|       >37F4 Logical record length                           |
|       >37F5 # of FIXED lenght records OR # of sectors for VARIABLE
|             length                                          |
|       >37F7 Reserved (>0000 >0000 >0000 >0000)              |
|       >37FF Pointer blocks                                  |
|       >38E3 Data Buffer area(256 bytes)                     |
|     ------------------------------------------------------------       |
|     >39E3 File Control Block for 2nd file OPENed (6 b)   s 518 bytes |
|                                                                        |
|       >39E9 File Desriptor record (256 bytes)                         |
|       >3AE9 Data Buffer area (256 bytes)                              |
|     ------------------------------------------------------------      |
|     >3BE9 File Control Block for 3rd file OPENed (6 b)     518 bytes |
|                                                                        |
|       >3BEF File Descriptor record (256 bytes)                        |
|       >3CEF Data Buffer area (256 bytes)                              |
|                                                              >3DEE    |
+----------------------------------------------------------------------+
|>3DEF          VDP STACK AREA                             252 bytes   |
|                                                                        |
|                                                              >3EEA    |
+----------------------------------------------------------------------+
|>3EEB          DISK DRIVE INFO                              4 bytes   |
|                                                                        |
|       >3EEB Last drive number accessed                                |
|       >3EEC Last track access on drive #1                             |
|       >3EED Last track access on drive #2                             |
|       >3EEE Last track access on drive #3                             |
|                                                              >3EEE    |
+----------------------------------------------------------------------+
|>3EEF        not used by the 4A , it might have been used     6 bytes |
|             by the 4 (?)                                              |
|                                                              >3EF4    |
+----------------------------------------------------------------------+
|>3EF5          VOLUME INFORMATION BLOCK                   256 bytes   |
|                                                                        |
|     An exact copy of sector >0 from the disk last accessed.           |
|                                                              >3FF4    |
+----------------------------------------------------------------------+
|>3FF5          FILE NAME COMPARE BUFFER                    11 bytes   |
|                                                                        |
|     Contains disk number & 10 character file name from last           |
|     access.                                                            |
|                                                              >3FFF    |
+----------------------------------------------------------------------+
```

References:
- -Millers Graphics, "THE SMART PROGRAMMER"
- -"MICROpedium"
- -Editor/Assembler's REFERENCE MANUAL
- -"TI-99/4A CONSOLE & PERIPHERAL EXPANSION SYSTEM TECHNICAL DATA"
- -'9900 FAMILY SYSTEMS DESIGN'
- -"TMS 9918A VIDEO DISPLAY PROCESSOR DATA MANUAL"

DISK DRIVE SPECIFICATIONS
VERSION 1.1, SEPTEMBER 12, 1985
by Louis Guion, Startext 77536

MANUFACTURER	MODEL NUMBER	HIGH	SIDE DENS	TPI	BYTES	5 V PWR	12V PWR	ACCES TIME	MOTOR DRIVE	COMMENT
AlpsElectric	FDD2225	1/2								
Canon	MDD211	1/2	DSDD	48	360K					
C.D.C.	9409	Full	DSDD	48	360K					
C.D.C.	9428	1/2	DSDD	48	360K				Drect	
Epson	SD521	1/2	DSDD	48	360K	.4A	.4A	6MSEC	Drect	O.K. in PBox
Hitachi	HFD505B	1/2								
Matsushita	JA551	1/2								
Matsushita	JA5551-2	1/2								
Micropolis	1115V	Full	DSQD							
Mitsubishi	M4851	1/2	DSDD	48	360K					
Mitsubishi	M4853	1/2	DSQD	96	720K	.5A	.7A			
MPI	851	Full	SSSD	48	90K				Belt	Sold in PBox
MPI	852	Full	DSDD							
MPI	501C-200	1/2								
MPI	502B-100	1/2	DSDD							
National	JA551-2	1/2	DSDD	48	360K					
Panasonic	JA551-2	1/2	DSDD	48	360K			6MSEC		
Qumetrack	142	1/2	DSDD	48	360K				Belt	Hi Pwr Reqmt
Qumetrack	142LX	1/2	DSDD	48	360K					
Qumetrack	542	Full		48						
Remex	RFD480	2/3	DSDD	48	360K				Drect	
Sanyo	FDA5200B/PC	1/2	DSDD	48	360K					
Sanyo	SM548D	1/2	DSDD	48	360K			6MSEC	Drect	
Shugart	400L	Full	SSSD	48	90K				Belt	Sold in PBox
Shugart	SA455	1/2	DSDD	48	360K	.6A	.6A	6MSEC		O.K. in PBox
Shugart	SA465	1/2	DSQD	96	720K					
Shugart	SA475	1/2		96	1.6M					For the "AT"
Siemens	FDD100-5	Full	SSSD	48	90K				Belt	Sold in PBox
Tandon	TM50-1	1/2	SSSD							
Tandon	TM55-2	1/2	DSDD	48	360K			6MSEC		
Tandon	TM55-4	1/2	DSQD	96	720K					
Tandon	TM65-2L	1/2								
Tandon	TM100-1	Full	SSSD	48	180K				Belt	
Tandon	TM100-2	Full	DSDD	48	360K				Belt	
Tandon	TM101-4		DSQD							
TEAC	FD55A	1/2	SSSD		180K			6MSEC		
TEAC	FD55B	1/2	DSDD	48	360K	.4A	.3A	6MSEC	Drect	O.K. in PBox
TEAC	FD55BV-06	1/2	DSDD	48	360K			6MSEC	Drect	No Hd Ld Sol
TEAC	FD55E	1/2	SSQD	96	500K			3MSEC	Drect	
TEAC	FD55F	1/2	DSQD	96	1M			3MSEC	Drect	
TEAC	FD55GFV-AT	1/2	DSQD	96	1.2M					For the "AT"
TEC	FB503	1/2	SSSD	48	90K				DRECT	2-O.K. in PBOX
Toshiba	5401		DSDD	-						
Toshiba	ND04D	1/2								
Toshiba	ND040T	1/2	DSDD							
Y.E.Data	YD580	1/2								

This information is intended to help TI-99/4A users in identifying disk drives that may be compatible with thier Peripheral Expansion Boxes and with their present disk systems. Since all information had been garnered from vendor advertisements, it is assumed to be correct, but must, none-the-less be used with caution due to transcription and other typographical errors.

If any reader can in any way add to the information presented, please do so by contacting the author at Startext MC 77536. Your help is appreciated!

FOR CASSETTE USERS:

Clyde Colledges High Speed Cassette Loader by Mickey Schmitt. As promised this month I am continuing with the topic of Clyde's loader. Let me say once again... if you are using a cassette system, THIS PROGRAM IS A MUST! It is by far one of the most impressive cassette utilities available to date!

While loading Clyde's program is not a difficult process in itself, understanding the procedure for the very first time can be a little confusing. With that in mind:

Instructions-

1. Insert Extended Basic Module
2. Select Option #2 - Extended Basic
3. Type: OLD CS1
4. Then: Press ENTER
5. Follow the directions as they appear on your monitor or TV screen.
6. Wait for the flashing cursor to appear in the lower left-hand corner of your monitor or TV screen.
7. Type: RUN
8. Then: Press ENTER
9. The computer will then return back to the Extended Basic screen with the message: *READY* and the cursor will once again be flashing in the lower left-hand corner of your monitor or TV screen. Clyde Colledge's High Speed Cassette Loader is now loaded.

INSTRUCTIONS FOR USING CLYDE'S LOADER:

1. After you have loaded Clyde's loader type: CALL LINK("OLD")
2. Then: Press ENTER
3. You can load in any program which you have on cassette in half the amount of time that it would have taken you normally!
4. Just follow the directions as they appear on your monitor or TV screen. That's all there is to it.

Clyde's loader has two very special features that should not go without mention. First of all... the high speed routines are exactly the same as Texas Instruments cassette routines; making this program very user friendly. Secondly... once the load program has been placed in the 32k memory, it will stay in memory... even if you accidentally hit "FUNCTION QUIT". Just retype: CALL LINK("ON") and you are ready to go. You can't lose the "Load Program" unless you turn off the console!

If you wish to purchase this program please send $5.00 (US) to:

WEST PENN 99"ERS
c/o John F. Willforth
R.D. #1 POX 73A
Jeannette, PA 15644

ATTN: WEST PENN LIBRARIAN

Editors note: This program ALSO allows cassette users to load and run programs LARGER than the 12k limit built into the console cassette routines. It DOES require Extended Basic and 32k memory... which you can install inside your console (refer to issue #5 of R/D Computing). The advantages of this utility are very obvious. Makes life with a cassette much easier!

Next month we promise to publish a schematic to make adding 32k even easier for console owners!

RENEWALS For many current 1986 subscribers this is the LAST issue of your paid subscription. If you have not renewed - do so now! Simply drop a cheque for $14.00 US or $20.00 CDN into the postage paid envelope and give it to your mailperson!

We will see you with VOLUME 2 ISSUE #20 next month. With the improved time frame we are now able to increase the size of this publication. Please note that we are at 20 pages starting with this issue.

Please help us by submitting articles, programs and interesting information for other TI owners around the world.

TOKENIZED COMMAND STORAGE by George F. Steffen

Some of you may have heard that there is a method of using a single key to enter a statement when programming. This is not an advertised feature of the TI 99/4(A) but results from the way TI Basic stores the program. Each statement in the program is stored as a single byte with a value over 127. The list of values and meanings is given below. HEX is the hexadecimal (base 16) value and DEC is the decimal value. Most of those values under 199 are available directly from the keyboard by the use of the Control key along with another key. When in the immediate mode, if you enter a number, the operating system assumes that you wish to enter a Basic line. If the line number is followed by a Basic statement, that statement is converted to its value and stored. If you enter a valid value, the conversion step is not necessary and the value is stored directly. However, if you then LIST the line, the meaning of the statement will be printed. For example, Control and Z equals REM; Control and U equals RANDOMIZE and Control and ; equals PRINT. You can experiment to find other keys which will equate to statements.

TOKENIZED COMMAND STORAGE

HEX	DEC	MEANING	HEX	DEC	MEANING	HEX	DEC	MEANING	HEX	DEC	MEANING
80	128	Note 1	A0	160	CLOSE	C0	192	>	E0	224	MIN *
81	129	ELSE	A1	161	SUB	C1	193	+	E1	225	RPT$ *
82	130	:: *	A2	162	DISPLAY	C2	194	-	E2	226	Note 1
83	131	! *	A3	163	IMAGE *	C3	195	*	E3	227	Note 1
84	132	IF	A4	164	ACCEPT *	C4	196	/	E4	228	Note 1
85	133	GO	A5	165	ERROR *	C5	197	^	E5	229	Note 1
86	134	GOTO	A6	166	WARNING*	C6	198	Note 1	E6	230	Note 1
87	135	GOSUB	A7	167	SUBEXIT*	C7	199	Note 2	E7	231	Note 1
88	136	RETURN	A8	168	SUBEND *	C8	200	Note 3	E8	232	NUMERIC*
39	137	DEF	A9	169	RUN *	C9	201	Note 4	E9	233	DIGIT *
8A	138	DIM	AA	170	LINPUT *	CA	202	EOF	EA	234	UALPHA *
8B	139	END	AB	171	Note 1	CB	203	ABS	EB	235	SIZE *
8C	140	FOR	AC	172	Note 1	CC	204	ATN	EC	236	ALL *
8D	141	LET	AD	173	Note 1	CD	205	COS	ED	237	USING *
8E	142	BREAK	AE	174	Note 1	CE	206	EXP	EE	238	BEEP *
8F	143	UNBREAK	AF	175	Note 1	CF	207	INT	EF	239	ERASE *
90	144	TRACE	B0	176	THEN	D0	208	LOG	F0	240	AT *
91	145	UNTRACE	B1	177	TO	D1	209	SGN	F1	241	BASE
92	146	INPUT	B2	178	STEP	D2	210	SIN	F2	242	Note 1
93	147	DATA	B3	179	,	D3	211	SQR	F3	243	VARIABLE
94	148	RESTORE	B4	180	;	D4	212	TAN	F4	244	RELATIVE
95	149	RANDOMIZE	B5	181	:	D5	213	LEN	F5	245	INTERNAL
96	150	NEXT	B6	182	)	D6	214	CHR$	F6	246	SEQUENTIAL
97	151	READ	B7	183	(	D7	215	RND	F7	247	OUTPUT
98	152	STOP	B8	184	&	D8	216	SEG$	F8	248	UPDATE
99	153	DELETE	B9	185	Note 1	D9	217	POS	F9	249	APPEND
9A	154	REM	BA	186	OR *	DA	218	VAL	FA	250	FIXED
9B	155	ON	BB	187	AND *	DB	219	STR$	FB	251	PERMANENT
9C	156	PRINT	BC	188	XOR *	DC	220	ASC	FC	252	TAB
9D	157	CALL	BD	189	NOT *	DD	221	PI *	FD	253	# (Files)
9E	158	OPTION	BE	190	=	DE	222	REC	FE	254	VALIDATE*
9F	159	OPEN	BF	191	<	DF	223	MAX *	FF	255	Note 1

Note 1. Meaning unknown, not used in Basic or Extended Basic.
Note 2. Unquoted string.
Note 3. Quoted string.
Both the above are followed by one byte giving the string length and then by the string. There is no closing quotation mark or end marker.
Note 4. Following two bytes are line number—second plus 256 times the first.
* Recognized by Extended Basic only.

ERROR CODE LISTING

EXTENDED BASIC ERROR CODES

10	Numeric overflow
14	Syntax Error
16	Illegal after Sbrtn
19	Name too long
20	Unrecognized Char
24	$/# Mismatch
28	Improperly used name
36	Image error
39	Memory Full
40	Stack Overflow
43	Next without For
44	FOR-NEXT Nesting
47	Must be in Sbrtn
48	Recursive Sbrtn CALL
49	Missing SUBEND
51	RETURN without GOSUB
54	String Truncated
56	Speech $ too long
57	Bad Subscript
60	Line not found
61	Bad Line #
62	Line too long
67	Can't CONtinue
69	Command Illegal in Prgrm
70	Only legal in prgrm
74	Bad Argument
78	No Program Present
79	Bad Value
80	Nil
81	Incorrect Argument List
82	Nil
83	Input Error
84	Data Error
97	Protection Violation
109	File Error
130	I/O Error
135	Sbrtn not found

EDITOR/ASSEMBLER ERROR CODES
XB ERROR EQUATES

ERRNO	>0200	2	Numeric Overflow
ERRSYN	>0300	3	Syntax Error
ERRIBS	>0400	4	Ill. after Sbprgm
ERRNQS	>0500	5	Unmatched Quotes
ERRNTL	>0600	6	Name too long
ERRSNM	>0700	7	$/# Mismatch
ERROBE	>0800	8	Option Base Error
ERRMUV	>0900	9	Improperly used name
ERRIM	>0A00	10	Image Error
ERRMEM	>0B00	11	Memory Full
ERRSO	>0C00	12	Stack Overflow
ERRNWF	>0D00	13	Next without For
ERRFNN	>0E00	14	FOR-NEXT Nesting
ERRSNS	>0F00	15	Must be in Sbprgrm
ERRRSC	>1000	16	Recursive Sbprgrm
ERRMS	>1100	17	Missing SUBEND
ERRRWG	>1200	18	RETURN without GOSUB
ERRST	>1300	19	String Truncated
ERRRBS	>1400	20	Bad Subscript
ERRSSL	>1500	21	Speech $ too long
ERRLNF	>1600	22	Line not found
ERRBLN	>1700	23	Bad Line Number
ERRLTL	>1800	24	Line too long
ERRCC	>1900	25	Can't Continue
ERRCIP	>1A00	26	Illegal in Program
ERROLP	>1B00	27	Only legal in Program
ERRBA	>1C00	28	Bad Argument
ERRNPP	>1D00	29	No Program Present
ERRBV	>1E00	30	Bad Value
ERRIAL	>1F00	31	Incorrect Argument List
ERRINP	>2000	32	Input Error
ERRDAT	>2100	33	Data Error
ERRFE	>2200	34	File Error
ERRIO	>2400	36	I/O Error
ERRSNF	>2500	37	Subprogram not found
ERRPV	>2700	39	Protection Violation
ERRINV	>2800	40	Unrecognized character
WRNNO	>2900	41	Numeric Overflow
WRNST	>2A00	42	String Truncated
WRNNPP	>2B00	43	No Program Present
WRNINP	>2C00	44	Input Error
WRNIO	>2D00	45	I/O Error

EXECUTION ERRORS

0-7	Standard I/O
08	Memory Full
09	Incorrect Statement
0A	Illegal Tag
0B	Checksum Error
0C	Dup. Definition
0D	Unresolved Ref.
0E	Incorrect Statement
0F	Program not found
10	Incorrect Statement
11	Bad Name
12	Can't Continue
13	Bad Value
14	Number too big
15	String/Number
16	Bad Argument
17	Bad Subscript
18	Name Conflict
19	Can't do that
1A	Bad Line Number
1B	FOR NEXT Error
1C	I/O Error
1D	File Error
1E	Input Error
1F	Data Error
20	Line too long
21	Memory Full
22	Unknown Error ode

****

LOADER ERROR CODI

0-7	Standard I/O
8	Memory Overflow
9	Not Used
10	Illegal Tag
11	Checksum Error
12	Unresolved Ref.

TI BASIC ERROR CODES PERTAINING TO DISK SYSTEM

#:	FIRST #	SECOND #
0:	OPEN	Can't find specified Disk Drive
1:	CLOSE	Disk or program is Write Protected
2:	INPUT	Bad Open Attribute
3:	PRINT	Illegal Operation
4:	RESTORE	Disk full or too many files opened
5:	OLD	Attempt to read past EOF
6:	SAVE	Device Error
7:	DELETE	File Error
9:	EOF	

TI WRITER ERROR CODES

0	- Indicates Disk Controller not on; OR: Diskette not Initialized
6	- No Disk in Drive; OR: Is upside down; OR: Drive is not turned on
7	- No Disk in Drive
00	- Illegal use of LoadF, PrintF: OR:
02	- No file in Diskette with Filename used
04	- Disk is full
06	- PrintF Command in progress was interrupted; OR: Disk Door was opened while Red Light was on
07	- Invalid Filename (I.E. Name too long or using invalid characters)
15	- Invalid Disk Drive Number, or Device

DISK MANAGER ERROR CODES

#:	FIRST #	SECOND #
1:	OTHER	Rec not found
2:	SEEK/STEP	Cyclic Redundancy
3:	INPUT	Lost Data
4:	PRINT	Write protect
5:	NIL	Write fault
6:	NIL	No Disk Drive
7:	NIL	Invalid input
8:	NIL	
9:	Special Error Code for Comprehensive Test	

I/O ERRORS

#	FIRST #	SECOND #
1:	OPEN	Device not found
2:	CLOSE	Write Protected
3:	PRINT	Invalid I/O Command
4:	RESTORE	Out of space
5:	OLD	EOF
6:	SAVE	Device Error
7:	DELETE	File/Data Mismatch

DISK MAP

by Earl Hall

The following is a complete and, to the best of my knowledge, accurate description of the Disk Directory format and file storage allocation used by the TI-99/4(A)
Earl Hall CompuServe ID - 72746,3244

SECTOR 0 - Volume Information Block

ESS	CONTENTS
0000-0009	Disk name - up to 10 characters
000A-000B	Total number sectors on disk (>0168=360, >02D0=720, >05A0=1440)
000C	>09 (# of sectors/trk)
000D-000F	'DSK' (>44534B) 44534B
0010	>50 = Disk backup protected, >20 = not protected
0011	# of tracks per side (>28=40, >23=35)
112-0013	# of sides/density (>0101=SS/SD, >0201=DS/SD, >0202=DS/DD)
)038-end	Sector allocation bit map. See note below

NOTE on >0038-end: This is a sector-by-sector bit map of sector use; 1=sector used, 0=sector available. The first byte is for sectors 0 through 7, the second for sectors 8 through 15, and so on. Within each byte, the bits correspond to the sectors from right to left. For example, if byte >0038 contained >CF00 then the first byte equals 1100 1111. This means that sectors 0 through 3 are used, sectors 4 and 5 unused and sectors 6 and 7 used. Information for the 2nd side of a DS/SD disk starts at byte >0065 and ends at byte >0091.

SECTOR 1 - Directory Link

Each 16-bit word lists the sector number of the File Descriptor Record for an allocated file, in alphabetical order of the file names. The list is terminated by a word containing >0000; therefore, the maximum number of files per disk is 127 [(256/2)-1]. If the alphabetical order is corrupted (by a system crash during name change, for instance), the binary search method used to locate files will be effected and files may become unavailable.

SECTOR >2 TO >21 - File Descriptor Records

ADDRESS	CONTENTS
0000-0009	File name - up to 10 characters
000C	File type: >01=Program(memory-image) >00=DIS/FIX >02=INT/FIX >80=DIS/VAR >82=INT/VAR File deletion protection invoked by Disk Manager 2 will be shown by >08 added to the above.
000D	# of (MAXRECSIZE) records/sector
000E-000F	Number of sectors allocated to the file. (Disk Manager 2 will list one more than this number, thereby including this sector in the sector count)
0010	For memory-image program files and variable-length data files, this contains the number of bytes used in the last disk sector. This is used to determine end-of-file.
0011	MAXRECSIZE of data file.
0012-0013	File record count, but with the second byte being the high-order byte of the value.
001C - end	Block Link (see note)

Note on file storage: Files are placed on the disk in first-come / first-served manner. The first file written will start at sector >0022, and each subsequent file will be placed after it. If the first file is deleted, a newer file will be written in the space it occupied.

If this space isn't big enough, the file will be 'fractured', and the remainder will be placed in the next available block of sectors. The block link map keeps track of this fracturing. Each block link is 3 bytes long. The value of the 2nd digit of the second byte followed by the 2 digits of the first byte is the address of the first sector of this extent. The value of the 3rd byte followed by the 1st digit of the 2nd byte is the number of additional sectors within this extent.

Sectors 2 through >21 are reserved for File Descriptor Records and are allocated for file data only if no other available sectors exist. If more than 32 files are stored on a disk, additional File than 32 files are stored on a disk, additional File Descriptor Records will be allocated as needed, one sector at a time, from the general available sector pool.

(reprinted from the newsletter of the Central Westchester 99'ers.)

FORMAT FOR DISK DIRECTORY/ALLOCATION OF FILE STORAGE
From: "The paper Peripheral" Central Texas 99/4A Users Group

The following is a complete and, to the best of my knowledge, accurate description of the disk directory format and file storage allocation used by the 99/4A computer.

SECTOR Ø CONTAINS THE VOLUME INFORMATION BLOCK

Address	Contents
ØØØØ-ØØØ9	Disk name--up to 1Ø characters
ØØØA-ØØØB	Total number of sectors on disk (>Ø168=36Ø, >Ø2ØØ=72Ø, >Ø5AD=144Ø
ØØØC	>Ø9 (# of sectors/trk)
ØØØD-ØØØF	'DSK' (>44534B)
ØØ1Ø	>5Ø=Disk backup protected, >2Ø=not protected
ØØ11	# of tracks per side (>28=4Ø, >23=35)
ØØ12-ØØ13	# of sides/density (>Ø1Ø1=SS/SD, >Ø2Ø1=DS/DD, >Ø2Ø2=DS/DD
ØØ38-end	Sector allocation bit map. See note below.

Note on >ØØ38-end: This is a sector-by-sector bit map of sec use; 1=sector used, Ø=sector available. The first byte is for secto Ø through 7, the second for sectors 8 through 15, and so on. With each byte the bits correspond to the sectors from right to left. For example if byte >ØØ38 contained >CFØØ then the first byte equals 11ØØ 1111. This means that sectors Ø through 3 are used, at byte >ØØ91.

SECTOR 1 CONTAINS THE DIRECTOR LINK

Each 16-bit word lists the sector number of the File Describtor Record for an alocated file, in alphabetical order of the file names. The list is terminated by a work containing >ØØØØ; therefore, the maximum number of files per disk is 127 [(125/2)-1]. If the alphebetical order is corrupted (by a system crash during name change, for instance), the binary search method used to locate files will be affected and files may become unavailable.

DISK ALLOCATION (CONT.)

SECTORS 2 TO 21 CONTAIN THE FILE DESCRIPTOR RECORDS

Address	Contents
0000-0009	File name--up to 10 characters
000C	Filetype: >00=DIS/FIX, >01=Program (memory-image), >02=INT/FIX, >80=DIS/VAR, >82=INT/VAR File deletion protection invoked by Disk Manager 2 will be shown by >08 added to the above.
000D	# of <MAXRECSIZE> record/sector.
000E-000F	# of sectors allocated to this file. (Disk Manager 2 will list one more than this number, thereby including this sector in the sector count.)
0010	For memory-image program files and variable-length data files, this contains the number of bytes used in the last disk sector. This is used to determine end-of-filo.
0011	MAXRECZIZE of data file.
0012-0013	file record count, but with the second byte being the high-order byte of the value.
001C-end	Block Link. See note below.

Note on file storage: Files are placed on the disk in first-come/first-served manner. The first file written will start at sector 0022, and each subsequunt file will be placed after it. If the first file is deleted, a newer file will be written in the space it occupid. If this space isn't big enough, the file will be 'fractured', and the remainder will be placed in the next available block of sectors. The block link map keeps track of this fracturing. Each block link is 3 bytes long. The value of the 2nd digit of the second byte followed by the 2 digits of the first byte is the address of the first sector of the extent. The value of the 3rd byte followed by the first digit of the 2nd byte is the number of additional sectors within this extent. Sectors 2 through 21 are reserved for File Descriptor Records and are allocated for file data only if no other available sectors exist. If more than 32 files are stored on a disk, additional File Descriptor Records will be allocated as needed, one sector at a time from the general available sector pool.

Fixing Blown Disks

TERRY ATKINSON

If you have had a disk drive for any length of time, chances are you have encountered such devastating messages as "disk not initialized" (when you know full well it is!), or "program not found " (when you know it is supposed to be there!). Or, perhaps, you have accidentally deleted a program and want to get it back. All of the above can be remedied.

FIXING THE DISK BIT MAP (AU0)

AU0, or Sector 0 contains the disk bit map, and if the characters "DSK" are altered, you will be unable to catalog or copy the disk. Indeed, a "DISK NOT INITIALIZED" error will show up. You can, however, retrieve programs and files individually and transfer them to another disk. That is, if you KNEW the names of ALL the programs/files on that disk. There is a better way which eliminates the possibility that you "forgot" about a particular program.

Boot up your disk fixer and load sector 0 from a disk. ANY disk will do. Then write the good sector 0 to the bad disk. This restores AU0 on the bad disk, but the bit map is NOT correct, but this does not matter. All you want to do is to be able to catalog and copy the disk using DM2. Use DM2 (not FORTH) to copy the entire disk to a new disk. You can then initialize the bad disk. That is all there is to it.

Ruined bit maps may not be discovered until it is too late. Any new programs saved to a disk with a ruined bit map may write over older programs or data. Goodby older program. There's nothing you can do about it.

Another possibility is that sector 0 has been damaged, perhaps by magnetism or a scratch on the surface. In this case, you'll find out when you try to read/write sector 0. You won't be able to. Now you have a problem, but not insurmountable. The only "fix" for this is to copy all sectors from the bad disk to a good disk, sector-by-sector. A tedious chore to be sure, but at least you can get all your programs back. It will still be necessary to proceed as above to get your programs back, as the bit map on the new disk will not be correct. Now, I am not sure how FORTH would behave under this circumstance. I know FORTH will "choke" when it tries to copy a damaged sector, but whether or not it will continue to copy the "good" sectors and put them into their proper places on the new disk, is beyond me. I wouldn't chance it. Better to be safe than sorry and stick to tried and proven methods. Of course, you could experiment. If it works, let us all know. If some of you FORTH addicts out there could shed some light on the subject, your comments would be most welcome.

FIXING THE DIRECTORY LINK MAP-(AU1)

S1 keeps track (alphabetically) of all the programs/files on the disk. Bad s1's could produce errors such that attempts to catalog the disk will produce a heading, but no programs, or maybe just "some" programs will be listed. To fix this, though, is extremely simp! Here's how:

First, look at AU0. Read the bit map to determin which sectors between 2 and 33 inclusive (>2->21) are flagged as used. Make a list of these sectors in a column. Now, load each of these sectors in turn, and examine the first 10 bytes of each. Copy the bytes down beside the relevant used sector. Determine the alphabetical order of these programs merely by readiing the numerical values. The lower the number, the closer to the front of the alphabet it is. Now, produce a list of these sectors arranged alphabetically. Here's a short example:

Sector used	Hex Values in 1st 10 bytes ())	Program Name
2	4B 20 20 20 20 20 20 20 20 20	K
3	49 20 20 20 20 20 20 20 20 20	I
5	4C 20 20 20 20 20 20 20 20 20	L
6	41 20 20 20 20 20 20 20 20 20	A
A	41 20 20 20 20 20 20 20 20 20	AB

Re-arranging the above alphabetically by sector would produce: 6,A,3,2,5 which are going to form the directory link map in WORD.

Next, copy sector 1 from ANY freshly initialized disk and write it to the bad disk. This is the easiest way to "restore" S1 to all zero's. Now, use the (A)lter command, and change the first, and each successive word to produ the alphabetical pointers. For example: 0006 000A 0003 00 0005 0000. Note the 0000 at the end. The directory link map must be terminated with the value, Now, write this sector to the bad disk, and you're in business.

RETRIEVING AN ACCIDENTALLY 'DELETED' PROGRAM

When you have a program in main memory, and type "new", the program is not erased. Only the pointers are changed, but the program is still in memory. A knowledgeable programmer could actually "unnew" a program, although not without difficulty.

The same applies if you "delete" a program from the disk. Only pointers are changed, and the program is still on the disk provided you have not performed a "save" since the deletion. Unlike main memory, retrieval of a deleted program from disk is extremely easy.

Locate the sector containing the deleted file's directory (between (>2->21). You can do this by using the "FIND STRING" command, or, if your disk fixer does not have this command, merely load them in one at a time and look for your "deleted" program's name in the first 10 bytes. Change the program name to "ZZZZZZZZZZ" (HEX code, of course). Now, write that sector back to it's proper spot. Load-in sector 1 and locate the first word containing 0000 replace it with the directory sector # of your dele program. Ensure the next word is 0000. Now, exit the DF and load the subject program as per normal. Exit the disk-fixer and load the program as normal and save it BACK to the same disk under the same program name (ZZZZZZZZZZ). Why? Because this will automatically update the disk bit map (AU0). Now use DM2 to change the program name back to it's original name and the task is complete.

A BRIEF ANNOTATED BIBLIOGRAPHY OF BOOKS RELATING TO THE TI-99/4A
(from the personal library of Barry A. Traver)

Assembly Language for the TI-99/4A

*Lottrup, Peter M.L. Beginner's Guide to Assembly Language on the TI-99/4A. Compute! Books, 1985. Although oriented toward Mini-Memory, this book is excellent for beginners, with very clear explanations and lots of short but useful program examples.

*McComic, Ira. Learning TI 99/4A Home Computer Assembly Language Programming. Prentice-Hall, 1984. A good book for beginners who have the Editor/Assembler but no previous experience in assembly language.

*Molesworth, Ralph. Introduction to Assembly Language for the TI Home Computer. Steve Davis Publishing, 1983. Primarily for use with the Editor/Assembler, but also can be used with Mini-Memory. Moves faster and further than the McComic book.

*Morley, M.S. Fundamentals of TI-99/4A Assembly Language. TAB Books, 1984. A good book for those who have the Mini-Memory Cartridge but not the Editor/Assembler.

BASIC Programs and Programming for the TI-99/4A

Ahl, David H. The Texas Instruments Home Computer Idea Book. Creative Computing Press, 1983. "Includes 50 Ready-to-Run Educational Programs," but most of them seem to be written in minimal BASIC and make no use of the special features of the TI-99/4A.

*Carlson, Edward H. Kids and the TI 99/4A. DATAMOST, 1982. This book is truly "not just for kids," but one of the *best* introductions to learning how to program in TI BASIC.

Casciato, Carol Ann, and Don Horsfall. TI-99/4A: 24 BASIC Programs. Howard W. Sams, 1983. Available with optional program cassette. Games, finances, home management, personal records, and utilities are included, all in TI BASIC.

*Compute!'s TI Collection: Volume One. A worthwhile collection of "over 30 TI-99/4A games, applications, utilities, and tutorials — most never before published," including a word processor, a data base management system, an electronic spreadsheet, several games, helpful programming tricks, and a super graphics program called "SuperFont."

Creative Programming for Young Minds...on the TI-99/4A. Creative Programming, 1982-1983. Several volumes in series. Hands-on instruction in TI BASIC (plus some small later reference to TI Extended BASIC). This series—like Carlson's book—is "not just for kids."

*Davis, Steve, ed. Programs for the TI Home Computer. Steve Davis Publishing, 1983. Four dozen programs that *do* make use of the special features of the TI-99/4A. Most of the programs only require TI BASIC and cassette system, though some make use of TI Extended BASIC, disk system, memory expansion, or Terminal Emulator 2 and speech synthesizer.

D'Ignazio, Fred. TI in Wonderland. Hayden Book Company, 1984. "21 programs for learning and fun," intended for youngsters, by the popular author of Katie and the Computer.

D'Ignazio, Fred. The TI Playground. Hayden Book Company, 1984. "23 programs for learning and fun," intended for young children.

Dusthimer, Dave and Ted Buchholz. The Tool Kit Series: TI-99/4A Edition. Howard W. Sams, 1984. Brief 5- to 15-line subroutines--dealing with color, sound and music, graphics, animation, and computation--that can be combined to form the basis of educational programs and computer games.

Engel, C.W. Stimulating Simulations for the TI-99/4A. Hayden Book Company, 1984. 11 "simulation game programs" in TI BASIC, 2 in TI Extended BASIC, adapted from a popular book first published in 1977.

*Flynn, Brian. 33 Programs for the TI-99/4A. Compute! Books, 1984. Although this book contains a few games, including a version of "Chomp" called "Vanilla Cookie," it is primarily concerned with mathematically-oriented programs, including money management and business programs, curve-fitting routines, matrix manipulations, statistics, and numerical analysis, all in Extended BASIC.

*Flynn, Christopher. Extended BASIC Home Applications on the TI-99/4A. Compute! Books, 1984. An excellent book containing data file management utilities, bar graph programs, an electronic card file, an appointment calendar, and two electronic spreadsheets. Flynn's programs always allow data to be saved on either tape or disk.

*Grillo, John P., and others. Data and File Management for the TI-99/4A. Wm. C. Brown Publishers, 1984. "Includes 48 programs to give the more advanced user techniques for information management." All programs are in TI Extended BASIC, and many make use of disk. Topics included: pointers, sorting, strings, linear and linked lists, sequential access files, direct access files, trees, and inverted files.

Grillo, John P., and others. Introduction to Graphics for the TI-99/4A. Wm. C. Brown, 1984. Includes 38 programs in TI Extended BASIC, some making use of disk, BUT note this comment by the authors: "In this book, we have limited our discussion to low-resolution graphics only. We do not discuss the color, sound, joystick, and lightpen features of this fine machine. We hope to cover these topics in a subsequent book."

Herold, Raymond J. TI-99/4A Sound and Graphics. A fairly good guide to sound, graphics, and speech synthesis on the TI-99/4A (including coverage of TI's text-to-speech diskette). Of the games, "Alphabet Invasion" and "Slot Machine" are done quite well.

Holtz, Frederick. Using & Programming the TI-99/4A Including Ready-to-Run Programs. TAB Books, 1983. Although this book is widely distributed, many chapters are either too elementary or too advanced to be of benefit to the average TI-99/4A owner.

Inman, Don, and others. Introduction to TI BASIC. Hayden Book Company, 1980. A straight-forward textbook on TI BASIC which does not go very far beyond the two manuals supplied with the TI-99/4A.

Knight, Timothy Orr. TI-99/4A Graphics and Sounds. Howard W. Sams, 1984. Available with optional program cassette. 37 sample (and simple) TI BASIC programs, originally written for the Commodore 64, most of which are rather trivial in nature.

Knight, Timothy Orr, and Darren LaBatt. TI-99/4A BASIC Programs. Howard W. Sams, 1984. Available with optional program cassette. Although these 30 TI BASIC programs were also originally written for the Commodore 64, they are more substantial than those contained in the other book by Knight.

Kreutner, Donald C. TI-99/4A Favorite Programs Explained. Que Corporation, 1983. 40 practical and entertaining programs in TI BASIC, with explanations.

*Loreto, Remo A., ed. The TI-99/4A in Bits and Bytes. Remo A. Loreto, 1983. A hodge-podge collection, but one containing within it a number of worthwhile programs (some in Extended BASIC) and programming hints.

Peckham, Herbert D. Programming BASIC with the TI Home Computer. McGraw-Hill Book Company, 1979. Another straight-forward textbook on TI BASIC, going a bit further than Inman's book.

Regena, C. BASIC Programs for Small Computers. Compute! Publications, 1984. Although this book contains "things to do in 4K or less" for other computers (notably the Vic-20 and TRS-80), it also contains programs in TI BASIC for the TI-99/4A.

Regena, C. Programmer's Reference Guide to the TI-99/4A. Compute! Publications, 1983. Not so much a reference guide as an instruction manual on how to program in TI BASIC, this book contains 48 programs by popular columnist Cheryl Whitelaw (or "Regena" of 99'er and Compute! fame).

Rugg, Tom, and others. 32 BASIC Programs for the TI-99/4A. dilithium Press, 1984. Programs include applications, education, games, graphics display, and mathematics. 30 programs in TI BASIC, 2 in TI Extended BASIC. (The programs can be ordered on disk or cassette.)

Sanders, William B. The Elementary TI-99/4A. DATAMOST, 1983. Contains useful chapters on "Data and Text Files" and "You and Your Printer," topics usually ignored in similar books.

Schechter, Gil M. TI-99/4A: 51 Fun and Educational Programs. Howard W. Sams, 1983. Available with optional program cassette. All programs are in TI BASIC, and all are probably 4K or less in size.

Schreiber, Linda M. and Allen R. The Last Word on the TI-99/4A. TAB Books, 1984. "55 practical and entertaining programs, all written in TI Extended BASIC," perhaps the best of which are "Battleship" and "Towers Game." (Programs are available on tape.)

*Sternberg, Charles D. TI BASIC Computer Programs for the Home. Hayden Book Company, 1984. Programs include automobile, conversion, home finances, kitchen helpmates, list, tutorial, and others, and each program is documented with description, symbol table, and output sample. The book is an adaptation for the TI-99/4A of Sternberg's BASIC Computer Programs for the Home; now if only someone will do an adaptation of his excellent two volumes on BASIC Computer Programs for Business!

Turner, Len. 101 Programming Tips & Tricks for the Texas Instruments TI-99/4A Home Computer. ARCsoft Publications, 1983. An unimpressive book carried in many bookstores.

Turner, Len. 36 Texas Instruments TI-99/4A Programs for Home, School & Office. ARCsoft, 1983. Many other books on this list contain a much better selection of programs in TI BASIC.

*Winter, Mary Jean. Computer Playground on the TI 99/4A. A colorful collection of TI BASIC computer activities intended for children in grades 2 through 6. Adapted for the TI-99/4A by Marcia Carrozzo.

*Wyatt, Allen. BASIC Tricks for the TI-99/4A. Howard W. Sams, 1984. Available with optional program cassette. A good collection of 28 useful subroutines dealing with selective input, rounding, dollars and cents, report formatting, time and dates, upper and lower cases, sorting, and menus.

*Zaks, Rodnay. Your First TI 99/4A Program. Like anything done by Zaks, this book is clearly written and well done. It is, however, ask the title indicates, a book for those who are just beginning to learn "the basics of BASIC."

Games in TI BASIC or TI Extended BASIC

Holtz, Frederick. TI-99/4A Game Programs. TAB Books, 1983. 32 "games, puzzles, and brain teasers" in TI BASIC, with explanations.

*Ingalls, Robert P. TI Games for Kids. Compute! Publications, 1984. An excellent collection of 32 educational game programs in TI BASIC for children ages 2 to 17.

McEvoy, Seth. Creating Arcade Games on the TI-99/4A. Compute! Publications, 1984. With the exception of one chapter devoted to TI Extended BASIC, this book tells "how to" write arcade games in TI BASIC, and includes eight finished games.

*Mullish, Henry, and Don Kruger. Zappers: Having Fun Programming and Playing 23 Games for the TI-99/4A. Simon & Schuster, 1984. Many favorites in TI BASIC, including "Blackjack," "Hangman," "Hidden Word Search," "Othello" ("Flip-a-Disk"), "Simon," and "Tic Tac Toe."

*Regena, C. TI Games. Compute! Publications, 1983. About 30 games for the TI-99/4A, mostly in TI BASIC, but including 7 in TI Extended BASIC, including the excellent "Mystery Spell" and "Mosaic Puzzle."

Renko, Hal, and Sam Edwards. Terrific Games for the TI 99/4A. Addison-Wesley Publishing Company, 1983. A mixed bag of 30-some unusual game programs from the Netherlands in TI BASIC and TI Extended BASIC.

*Singer, Scott L., and Tony E. Bartels. Games TIs Play. DATAMOST, 1983. 32 TI BASIC game programs based on the book Games Apples Play by Mark James Capella and Michael D. Weinstock. (Programs are available on disk.

*Ton, Khoa, and Quyen Ton. Entertainment Games in TI BASIC and Extended BASIC. Howard W. Sams, 1983. Available with optional program cassette. One of the *best* program collections available; "Frogger"-lookalike "HomeBound" is excellent. Book also contains a few non-game programs, e.g., "Address Inventory" and "Auto Sprite Editor."

LOGO Programs and Programming for the TI-99/4A

*Abelson, Harold. TI LOGO. McGraw-Hill Book Company, 1984. If you have TI LOGO II, you already have this excellent book, but if you have TI LOGO (I), get it!

Bearden, Donna. 1, 2, 3, My Computer & Me. Prentice-Hall, 1983. Though not just for the TI, this "LOGO funbook for kids" contains an appendix on "editing features for Apple LOGO, MIT LOGO, and TI LOGO."

*Conlan, Jim, and Don Inman. Sprites, A Turtle, and TI LOGO. Prentice-Hall, 1984. "A friendly, playful introduction to the TI LOGO computer language," very well done.

*Programming Discovery in TI LOGO. Texas Instruments, 1982. This attractive "student guide" was used by Texas Instruments with their Computer Advantage Clubs and is very well designed.

Ross, Peter. Introducing LOGO: For the Apple II Computer, Texas Instruments 99/4A, and Tandy Color Computer. Ross comments that "TI LOGO differs from Terrain LOGO and Apple LOGO in several important ways.... The main difference is that TI LOGO has 'sprites' and 'tiles' as well as the turtle." TI LOGO II also has music. Ross's book is useful, but unspectacular.

Thornburg, David D. Computer Art and Animation: A User's Guide to TI-99/4A Color LOGO. Addison-Wesley Publishing Company, 1984. This book is also an introduction to TI LOGO, more general in content than the title might suggest.

*Watt, Daniel. Learning with LOGO. McGraw-Hill, 1983. Although primarily concerned with Terrapin/Krell LOGO and secondarily with TI LOGO, this is one of the best and most comprehensive books on LOGO presently available.

Miscellaneous Books for the TI-99/4A

*The Best of 99'er: Volume 1. Emerald Valley Publishing, 1983. A very worthwhile collection of articles on "Starting Out," "Programming Techniques and Languages," "Inside BASIC and Extended BASIC," "LOGO," "Assembly Language," "Computer-Assisted Instruction," "Computer Gaming," and "Applications and Utilities."

Blackadar, Thomas. The Best of TI 99/4A Cartridges. SYBEX, 1984. As the title indicates, this book only covers some of the cartridges (but, in my opinion, not always the best). Nevertheless, this is one of the few books that has any significant treatment of cartridges for the TI.

Brewer, Bill. The TI-99/4A User's Guide. Macmillan, 1983. How can you not like a book whose cover blurb says this?: "There is only one home computer priced below $100 that has a microprocessor as powerful as the expensive IBM PC's. And that home computer has more educational cartridges produced for it than for any other system. It's the TI 99/4A, the best computer value for its price on the market today."

*Casciato, Carol Ann, and Donald J. Horsfall. The TI-99/4A User's Guide. Howard W. Sams, 1983. An excellent book, carefully done, by two authors who know the TI-99/4A well.

Garrison, Paul. The Last Whole TI 99/4A Book: Programs and Possibilities. Wiley Press, 1984. Contrary to the promises on the cover, this is not "the only book you need," although it does cover a lot of ground (with a few inaccuracies here and there).

*Heller, David and Dorothy. Free Software for Your TI-99/4A. Although the information is not always entirely accurate, this book contains much information not readily available elsewhere.

Micronova's Home Computer Directory for the TI 99/4(A). Micronova, 1983. A very useful book when it first appeared, although some of the information is now significantly dated.

The User's Guide to Texas Instruments TI-99/4A Computer, Software, & Peripherals. Beckman House, 1983. A useful guide "by the editors of Consumer Guide," this book has appeared in several different formats.

Willis, Jerry, and others. Things to Do with Your TI-99/4A Computer. New American Library, 1983. Part of a series prepared by dilithium Press, this book is fairly competent as an outside look, but unimpressive.

Albright, Ron. The Orphan Chronicles. Millers Graphics, 1985. A history of the TI Home Computer and sources of information about it.

*Especially recommended.

This list (prepared by Barry Traver, 835 Green Valley Drive, Philadelphia, PA 19128) is not complete, but should prove useful to those who are interested in knowing more about some of the books that are available for the TI-99/4A.

A Description and Commentary on the Geneve Computer
Some Implications for us all

by Chris Bobbitt
President, Asgard Software

At its introduction, the Myarc Geneve computer will be among the most advanced computers available, and definitely the most advanced "home computer" in history. It is more powerful than many minicomputers, but is available at a price that would have been unheard of 3 years ago.

The following is a description of some of the capabilities of this remarkable device.

MICROPROCESSOR:

The TMS9995 CPU is 5 to 6 times faster than a TMS9900, the processor found in the TI99/4A. This processor is only slightly slower than the 68000 CPU, yet is much simpler to use, more accurate mathematically, and contains a smaller instruction set. The advantages of this smaller instruction set is an article in itself. Suffice it to say that this technique, called RISC, is getting a lot of attention in programming circles.

MEMORY:

The standard Geneve Computer comes with 640K of RAM. This is expandable to 2 Megabytes using special memory expansion devices. A Myarc 512K card can be made to work with the Geneve with simple modifications. The Myarc 512K card memory may be directly accessed by programs.

GRAPHICS

The Geneve uses the Yamaha 9938 graphics processor. The 9938 processor was designed by Texas Instruments and Microsoft Incorporated. The computer world will discover this chip and its capabilities much in the same way that they proudly announced 16 bit computing for microcomputers, years after TI had introduced the TI99/4A. This graphics processor supports a variety of different modes for graphics and text.

TEXT

The Geneve supports both 40 AND 80 column modes. The 40 column mode is similar to that of the 99/4A, so none of your current word processing software is obsolete. However, text, foreground and background colors may be any of 512 colors. 256 patterns are available for redefinition. One of the 80 column modes is the same, while another supports blinking text and multi-color text. Some limitations apply, but this permits programmers of the system to use many of the advanced human factors graphics techniques just now being developed. The use of color to impart information, much in the nature of peripheral vision can make word processing tasks was well as the initial learning process easier. Your Geneve computer will be able to keep up with this emerging technology for some time. Indeed the rich resources of the TI programming community may well result in some breakthroughs in graphics presentation. It is reasonably well known that some organizations in the community are working hard in this area. Since each of these various screens, occupies very little memory of the 128K of standard video RAM found on the Geneve, up to 32 screens of text can be stored in memory at once. All of this information is directly addressable by the programmer. This bodes well to provide a rich environment for the system and applications programmer and thus the user.

The Geneve supports every text mode of the 99/4A, as well as many new modes that use much of the available memory. One of the more interesting modes supports a resolution of 256 by 216 pixels. Each pixel can be any of 256 colors. This mode also supports multi-color sprites. Each pixel row of the sprite can be any of two colors. Another interesting graphics mode supports 512 by 424 pixels with each pixel any of 16 colors. The on-screen display of a maximum of 16 different colors can be selected from a pallet of 512 colors. This mode is the same resolution as the Apple MacIntosh computer, yet the system still finds the capabilty to support sprites, which the MacIntosh does not. The 9938 chip has built in commands for line drawing, block moves and copies at hardware speeds. Programmers will have a rich, challenging environment for creativity, all at an affordable price for 99/4a owner and convert alike.

INTERFACES

The Geneve has a number of ports. For video, there is a port for an analog RGB monitor. The analog RGB monitor is more advanced than the digital ones used by the TI Professional Computer. Texas Instruments used the quality of the TI PRO monitor as a major component in its "Dare to Compare" campaign against the inferior IBM PC display system. An Amiga monitor displays the power of the Geneve quite well, and is readily available. However, an additional port permits the use of your existing TI99/4A video monitor. Therefore, your current equipment is not obsoleted by the new machine, allowing you the luxury of leisurely getting the best price for your existing monitor and cutting the best possible deal for your upgrade. Indeed, some are already at work seeking to separate early dropouts in the Amiga world from their monitors. The Geneve also supports the Amiga mouse. Other monitors of the serial RGB type work, however, so do not pay extra simply because the name on the front.

Your 99/4A console can be used as a stand alone device with the purchase of the Geneve. The Geneve comes equipped with an IBM style keyboard. Other keyboards, costing from $50 to $500 will also work just fine. Since the Geneve replicates the functions of the console, you will only need the expansion system or one of the inexpensive expansion kits.

A multifunction port permits even more access to the Geneve. While labeled as being for the Amiga mouse mentioned earlier, also can support sophisticated applications input from equipment both exotic and common. A video digitizer, for instance. Pictures taken from a video camera can be fed into the system. A digitizing tablet, which turns the Geneve into an elaborate data collection system or a component of a computer aided design (CAD) sytem is fully supportable, given proper software. Light pens are of course appropriate input devices as is information from a video cassette recorder or a video camera. Indeed, with external converter devices available on the market, you can pipe in television signals and enjoy crisp resolution and vibrant colors never seen before from a commercial television set, thus putting your RGB monitor on overtime.

HARDWARE COMPARISONS

To put this in perspective, compare the Geneve to other computers. The Geneve comes with 640K of RAM, equivalent to a fully configured IBM PC XT. This memory is expandable to 2 megabytes, twice the standard memory of an Atari 1040 ST. The Atari ST, of course, is one of the more popular "non IBM machines" on the market. The Atari ST is the fastest microcomputer available in its price range. The Geneve is roughly equivalent. The makers of the Geneve have gone to the extra expense of installing special purpose chips to handle, among other things, input from disks, lightpens, and other devices. In a similar vein, these special purpose chips handle output to screen, disk and elsewhere. And what about graphics? Again expensive special purpose redundance pays off. Therefore, in graphics, input and output, the Geneve runs circles around the Atari ST. The Geveve deploys eight times as many colors as the Commodore Amiga. The Amiga is the superior machine in these respects. The Geneve, unlike the Amiga and the IBM PC AT, supports graphics with a 'true aspect' ratio. This is the superior form, and gives higher resolution through the use of square pixels, the tiny dots used to give your computer screen, even your television its color and appearance of depth.

The Geneve rates highly as a smoothly upgradeable machine. It obviously will be compatable with the newly developed Myarc disk controller card. In disk drives supported, the Geneve with the Myarc disk controller card will defeat the IBM PC AT. Four 20 megabyte hard disks can be supported with this upgraded configuration, not to mention that the same scheme will control four (or less) double sided QUAD density floppie drives of the conventional 5 1/4 inch size. The drives that use the new plastic bound three inch disks are supported as well. Knowing the market, the Geneve makers realised they needed a system that would obsolete gracefully, as has the 99/4A.

Features of the 99/4A which still challenge the marketplace are retained. An example is the 99/4A's well known device independant operating system. Virtually any peripheral can be attached, unlike almost all other computers including those costing thousands. Device independence is a feature you (the 99/4A owner) have purchased years ago and one that should not be discarded in the name of progress. Therefore, the Geneve is superior to most every micropcomputer in graphics, speed, memory capacity, and in versatility.

A full blown Geneve system would contain a Geneve computer, a WDS model hard and floppy disk controller, a TI RS232 card, plus a 3 slot expansion kit, linked to two full blown 720 kilobyte floppy disk drives and a high resolution serial RGB monitor. If bought all at the same time, using all new components, your system would cost less than $1,000. One of the finest features of such a system is that it can and probably should be acquired incrementally, particularly if you currently own an expanded 99/4A system. For a machine of this class, this is an incredible price. The Atari 1040 ST is well known as the first computer that cost less than One dollar for each one thousand bytes of memory, new. The Geneve may be the first machine to drive that cost down to fifty cents per thousand.

SOFTWARE

The Geneve will come bundled with a new version of Extended BASIC on disk which is fully 6 times faster than TI Extended BASIC. Also included will be a MS-DOS like operating system. The package is called "DOS like" because the commands used will be very close to MS-DOS. However, the internal workings of the system will not resemble nor be compatable with MS-DOS. This will be a boon for those who have had to struggle through learning MS-DOS at work or on another machine. In the package also will be an 80 column version of TI-Writer with a larger memory.

A number of other products specifically designed for the Geneve will be available at or near the release of the Geneve. A number of 'C' compilers will be available by all expectations. C is a very popular language on 32 bit machines and is now beginning to appear in micro computers in the last few years. Some business software will be readily available. UCSD Pascal, actually a language within its own operating system, will also be standard. Software developed on many machines, including the IBM PC, Apple, and others which use this system will run without modification on the Geneve.

The new Geneve software will allow users to set up directories as an aid to manage multiple files. A software RAMdisk will also be available, where the user can deal with a notional or in-software emulation of a disk. All interaction on this RAMdisk will be in memory, thus will operate at extremely high speed. Print spoolers will be available. People still pay $200 for print spoolers, which merely are hardware systems, now software, that fool both the computer and the printer. The printer is wired to signal the computer to stop sending data while the printer repositions the print head, or rolls up the platen. Meanwhile the computer is burning up thousands of cycles waiting for printer to get ready to receive data again. A spooler is nothing but an ever ready printer to the computer and a patient computer to the printer. The job is transmitted to the spooler in a second or two and you are ready to go again while the printer chunks away.

TI BUSINESS MACHINES-The Geneve is assembly language compatable to the TI mini computer world, and awaits a member of that community to make that software run.

There is one silver lining in the "Perils of Pauline" development path of the Geneve, so fraught with delays. Time to think about the new arrival has been purchased with the sweat of the developer in a process which would normally have been extremely secret and quickly sprung on the unsuspecting community with little warning.

NEW OFFERINGS

One new company has been started specificly to develop Geneve software. A true multi-tasking operating system is among the goals of this firm. Multi-tasking to a user means that several programs can be run at the same time. Multitasking is at the heart of such programs such as Sidekick for the IBM where various panels, or windows are pulled down to allow notes and other activities to take place.

Yet another goal for this new developer is a macro-assembler. Macro-assemblers are small utility programs that can be strung together to achieve a variety of goals. In the mini computer world, programmers adroit in the macros of their particular machine rarely had to write much original code to achieve powerful results. This capability will soon arrive for you with the Geneve.

Soon after shipments of the Geneve begin, BASIC and Pascal compilers will be made available by this startup firm. A compiler may not be a familiar concept to all who read this, though it is simple to pick up. When your 99/4A receives the run command, it wakes up and "interprets" the program you have told it to run; Every single time. You probably are aware that assembly language is faster. The reason for this is that it is closer to machine language and therefore requires minimal "interpretation." BASIC, however, along with a host of other languages is not that close to machine language. Easier to remember and use, but requiring some form of intervention. The interpreter is often used for BASIC. While it gives instant feedback, an interpreter is slower than a compiled program which is a machine or assembly language program. You write the program as usual, then run the program through a compiler. That program compiles a collection of assembly language or machine code commands. That "compilation" is what you then use when you need that program. The compilation is much faster, almost indistinguishable from a program written in assembly language. The 99/4A only recently got an example of a compiled BASIC and a compiled C. If you have yet to experience the utility of compilers, you will certainly enjoy the Geneve. The increased memory will, of course, make these compilers superior in performance to anything currently on the 99/4A.

YET ANOTHER HUGE LIBRARY-Not one but two major resources are in the game plan for this firm. CP/M is an operating system that has its own cult following, and is still supported by a major commercial and cottage industry. Transfer of CP/M (and yes, IBM) disks to the Geneve is in the works. The firm is called Access Engineering, and is located in the Washingtn D.C. area.

A HOST OF GENEVE SPECIFIC PROGRAMS are to come. Lou Phillips of Myarc has estimated that four to five years of effort will be needed to complete the full sweep of programs needed to truely tax the Geneve system and the chips associated with it. During that period, if a new design comes along, the card, not the entire structure can be modified. Almost immediately however, terminal emulators, word processing programs that support such sophisticated typesetting concepts as proportional spacing will begin to arrive.

"Goodbye 'til we meet . . ."