WEST JAX 99'ERS NEWS

Dedicated to the {ORPHAN?}

OCTOBER 1986 #19

The West Jax 99'ers is a non-profit computer users group for the TI99 Home Computer. (Not affiliated in any way with Texas Instruments.) The club's mailing address is PO BOX 176, Orange Park, FL 32067

MEETINGS - Second Tues. of every month at the Webb Library, two lights west of Blanding on 103rd Street. Workshop time on the club computer begins at 6 PM, business meeting at 7 PM, more workshop time afterwards. Visitors are welcome to attend and find out more about the club activities and library. A second meeting of strictly workshop time is held on the Fourth Tuesday of the month beginning at 6PM.

OFFICERS -	President	Rick Felzien	(904) 772-9162
	Vice-Pres.	Steve Peacock	(904) 737-2859
	Secretary	Ralph Glatli	(904) 757-3630
	Treasurer	Thomas LeMay	(904) 282-5220
WEST JAX	TIBBS BBS	Art Pugh(SysOp)	(904) 272-8067

For newsletter submissions and suggestions, contact Rick Felzien.

****
****

To all the new members, a hearty welcome aboard and a sincere wish for your continued participation in the Users Group. Although we the older members seem at times to neglect you, we do all appreciate your presence and hope to see you often.

The Logo article this month has what the author hopes is an interesting couple of programs. As I continue to delve into Logo in the quest for information, the more I realize that, although Logo is slow, it is very powerful yet easy to program.

There is also an article from the Logo author on the TI-99/4A itself which should be of interest to those of you who are new to the TI and possibly know what a powerful machine it is, but don't know why.

****
****

```
*************************************
Number: 353  Name: FASTTERM-FIX
Address: BDHALL  Date: 860806
Approximate # of Bytes: 2520
Number of Accesses: 69  Library: 2
*************************************
Description:
Fix for FastTerm for downloading
long XMODEM files.  In ASCII.
Keywords: FastTerm, FIX, ASCII
-------------------------------------
```

Fast Term Fix

Thanks to Howie Rosenberg I finally got the correction to Fast Term I was looking for. I inserted the code from byte >82 to >99 instead of >74 to>8A as that's where I found it. The program was tested at both 300 && 1200 baud and the downloads worked fine. Maybe our problem is solved. I would suggest using a backup copy to play around with rather than your good one:

Here is a simple patch to Fast-Term V1.16 to allow it to download files longer than >40 records from those BBS which didn't used to work... edit the second program image file, (TRM2, for most of you), sector #22 (21 if you start from 0) start at byte >74:

```
addr^:^should have^^^^:^^change to
>74^^^^>C020^^^^^^^^^^^^^^>0221
>76^^^^>CCCE^^^^^^^^^^^^^^>FF00
>78^^^^>1303^^^^^^^^^^^^^^>9801
>7A^^^^>0201^^^^^^^^^^^^^^>D0B3
>7C^^^^>CFDA^^^^^^^^^^^^^^>1603
>7E^^^^>10E1^^^^^^^^^^^^^^>0620
>80^^^^>0221^^^^^^^^^^^^^^>D0B2
>82^^^^>FF00^^^^^^^^^^^^^^>1003
>84^^^^>9801^^^^^^^^^^^^^^>0201
>86^^^^>D0B3^^^^^^^^^^^^^^>CFDA
>88^^^^>16F8^^^^^^^^^^^^^^>10DC
>8A^^^^>0202^^^^^^^^^^^^^^>0202
```

I found on my version of Fast-Term V1.16 that the change was in the same sector as mentioned above but the code change was slightly different. In mine it looked like this:

addr :	should have :	change to
>82	>C020	>0221
>84	>CCD0	>FF00
>86	>1303	>9801
>88	>0201	>D0B3
>8A	>CFDC	>1603
>8C	>10E1	>0620
>8E	>0221	>D0B2
>90	>FF00	>1003
>92	>9801	>0201
>94	>D0B5	>CFDA
>96	>16F8	>10DC
>98	>0202	>0202

Here's how you can do it using D-Patch in Funlwriter. Put TRM2 on a freshly initialized disk, load D-Patch, look at sector >37. Go to line 10, the nineth character and start to enter the code in the other message, i.e., 0221 FF00 9801 D0B3 ...etc. When you get to the 2020 at the end you should find that you are at the end of row 11. Now save the sector back to disk (FCTN 8) and you're ready to try it. Now copy the rest of Fast Term onto that disk and make sure it works!. This is a test to see if you have been reading my Odds & Ends articles about sector editors.

T H E   B A S I C   A S S E M B L E R #14 By Steve Peacock

SPEECH

Creating speech with assembly is fairly complicated. However like all difficult things it gives you great satisfaction when done. I must take time to give a hearty THANKS to Mr. Nick Iacovelli Jr. for his help with the writing of this assembly program. I was unable to understand the Editor/Assemble manual on how to write a speech program. He showed me how. Thanks Nick!

Start your code with a reference to speech write and read (SPCHWT SPCHRD). As you can see from this months assembly code, some data and other information must be set at the start. Do not forget to add LIMI 0 and LIMI 2, so the 'FCTN' quit key will work.

Any word that is in the resident vocabulary may be spoken. This list is on pages 422 to 427 of the Editor/Assembler manual. To speak a word you must find it's address from this list. The word 'COMMAND' has an address of >1F1A. This is four nibbles, 1, F, 1, and A. Each of these nibbles must be written to register one. They are written backwards so for the word 'COMMAND' they are written A, 1, F, and 1. You must also add 4x to each. This make the four nibbles: 4A, 41, 4F and 41.

As the last nibble you must add 40, and write it to speech write. The command H50 is then given. This is what causes the computer to speak. As the computer is speaking you must check to see if it is done. If it is not then jump to a time delay. COC @H8,R0 checks to see if done. If not, jump to a delay and then return to check again.

After the word is spoken, the program jumps back to the start and is in an infinite loop, and the word is spoken again.

As you can see, quite a bit of coding is need in assembly to say the word 'COMMAND'. Only one line is needed in Extended BASIC. Where is the advantage? Time. If you listen to the assembly version and the BASIC version, you will be able to tell that the assembly program is running faster.

###

```
100 REM PROGRAM BA14B==>Basic Assembler #14 BASIC Version
110 REM SPEECH
120 REM (C)1986 S. PEACOCK
130 REM YOU MAY WANT A 'CALL CLEAR' HERE
140 CALL SAY("COMMAND")
150 GOTO 140
160 END
```

###

```
*******************************************************************************
*
* PROGRAM BA14==>Basic Assembler  #14 Assembly Version
* SPEECH
* (C)1986 S. PEACOCK
*
*******************************************************************************
* THANKS TO MR. NICK IACOVELLI JR. FOR HIS HELP WITH THIS PROGRAM.
       DEF  START
       REF  SPCHWT,SPCHRD     *REFERENCE TO SPEECH WRITE AND READ
HHROM  DATA >0000
H50    BYTE >50
HAA    BYTE >AA
H8     BYTE >80
       EVEN
START  LIMI 0
       LIMI 2
* THE WORD TO SPEAK IS 'COMMAND'
* ADDRESS IS >1F1A
* EACH NIBBLE IS PUT IN FROM RIGHT TO LEFT
* ADD 4x TO EACH NIBBLE => 4A, 41, 4F, AND 41
       LI   R0,>4A00          *FIRST NIBBLE 'A'
       MOVB R0,@SPCHWT        *WRITE IT
       BL   @DLY12            *WRITE DELAY OF 12 MSEC
       LI   R0,>4100          *SECOND NIBBLE '1'
       MOVB R0,@SPCHWT
       BL   @DLY12
       LI   R0,>4F00          *THIRD NIBBLE 'F'
       MOVB R0,@SPCHWT
       BL   @DLY12
       LI   R0,>4100          *FOURTH NIBBLE '1'
       MOVB R0,@SPCHWT
       BL   @DLY12
       LI   R0,>4000          *'40' MUST BE ADDED AS A LAST NIBBLE
       MOVB R0,@SPCHWT
       BL   @DLY12
       MOVB @H50,@SPCHWT      *HEX 50 IS COMMAND TO SPEAK
WAIT   BL   @READIT
       MOV  @SPDATA,R0
       COC  @H8,R0            *CHECK TO SEE IF SPEECH IS DONE TALKING
       JEQ  WAIT              *IF LEFT OUT MACHINE MAY LOCK UP!
       JMP  START
       AORG >8328
SPDATA DATA 0
READIT MOVB @SPCHRD,@SPDATA
       NOP                    *DELAY
       NOP
       NOP
       RT
DLY12  NOP                    *DELAY 12 MSEC
       NOP
       RT
DLY42  LI   R1,10             *DELAY 42 MSEC
DLY42A DEC  R1
       JNE  DLY42A
       RT
       END
```

TEXT PROCESSING WITH LOGO

(With programs and excerpts from issues of 99'ER)

BY

Rick Felzien

(The Logo Logician)

When we talk about word processing with Logo we mean the creation and formatting of a text document such as a letter for output to a printer. As those of you who use their TI-Writer a good bit know, being limited to a screen of less than 80 columns is no real limitation in the preparation of well formatted output. The most important thing is a good set of screen editing commands and the ability to format the text to your particular desires.

The fomulation of text is best accomplished using Logo's own built-in-editor. Since our main interest is printing the text that we have created, and since Logo can only print procedure definitions, we have to trick Logo into thinking that we are defining a procedure. The first thing that we must do then is have a first line such as-TO (pocedure name), with with "procedure name" bein the name of our text file. Then all we have to do is use the normal Logo editing keys and commands to create the text file. When we are finished with our text we press FCTN 9(BACK) and our text will appear to Logo to be a defined procedure.

Let's write a sample text file and call it DOC:

```
TO DOC
THE NUMBER OF SCREEN COLUMNS
ISN'T PARTICULARLY IMPORTANT
IN TEXT PROCESSING. HOWEVER.
IT IS VERY IMPORTANT TO BE
ABLE TO FORMAT OUTPUT FOR A
GIVEN LINE WIDTH.

REFORMAT IS A PROCEDURE FOR
REFORMATTING OTHER
PROCEDURES. ITS INPUTS ARE
THE OLD NAME
THE LINE WIDTH, AND THE
NEW NAME
THE NEW NAME CAN BE THE SAME
AS THE OLD NAME.
END
```

When Typing in the EDIT mode, we notice that we can keep typing without hitting ENTER until 127 characters have been entered. The text will wrap similar to WORD WRAP in TI-Writer. Logo does not, however allow for indenting as seen above. To indent we must use a non-space character.

In addition to having ease of text creation, we must be able to easily reformat to ay specifications for output to a printer, to have a good word processor.

REFORMAT will reformat a document to any width we chose. Its specification is, REFORMAT (old procedure-name) width (new procedure-name). For example, using our created text, REFORMAT "DOC 25 will rformat our text to 25 columns and store the result in a new file called DOC25. To print out your file just RECALL it and use the PP command. The following procedure can be saved on the files disk and loaded to get rid of unwanted files in memory.

```
TO CLEANUP
ERASE REFORMAT
ERASE REFORMAT1
ERASE FLUSHBUFF
ERASE PUTINBUFF
ERASE EMPTYBUFF
ERASE LENW
ERASE CLELANUP
END
```

How does REFORMAT work? It uses a few special Logo primitives TEXT and DEFINE. TEXT converts the text of a procedure definition into a list. DEFINE converts the list of a procedure into a procedure. REFORMAT's job is to pass the list representation of the of the input procedure definition to REFORMAT1 and take REFORMAT1's output and convert it to a procedure. The main work is done by the REFORMAT1 procedure

There are six inputs to REFORMAT1. The first two are the list of lines to be processed and the desired width. The other four are initial values for local variables.

So, as you will see after you type in this procedure and run it, TI-LOGO is a great deal more powerful than many people realize with their first encounter with the package.

The following is the listing for the Logo Word Processor.

```
TO REFORMAT :TNAME :LL :NTNAME
MAKE "T TEXT :TNAME
MAKE "T REFORMAT1 :T :LL [ ] [ ] 0 [ ]
DEFINE :NTNAME :T
MAKE "T [ ]
END

TO EMPTYBUFF
IF :BUFF = [ ] THEN STOP
PRINT :BUFF
MAKE "NTXT LPUT :BUFF :NTXT
MAKE "BUFF [ ]
MAKE "BL 0
END

TO REFORMAT1 :TXT :LL :CURL :BUFF :BL :NTXT :1
IF :CURL = [ ] THEN GO "2
MAKE "W FIRST :CURL
MAKE "WL LENW :W
TEST :BL + :WL > :LL
IFT FLUSHBUFF
IFF PUTINBUFF
GO "1
2:
IF :TXT = [ ] THEN EMPTYBUFF OUTPUT :NTXT
MAKE "CURL FIRST :TXT
TEST :CURL = [ ]
IFT EMPTYBUFF FLUSHBUFF
MAKE "TXT BF :TXT
GO "1
END
```

```
TO PUTINBUFF
MAKE "BUFF SE :BUFF :W
MAKE "BL 'BL + :TXT
GO "1
END

TO FLUSHBUFF
PRINT :BUFF
MAKE "NTXT LPUT :BUFF :NTXT
MAKE BUFF [ ] .
MAKE "BL O
END

TO LENW :W
IF FIRST :W = :W THEN OUTPUT 1
OUTPUT 1 + LENW BF :W
END
```

The following is a very interesting game in Logo. It is an adventure, of all things. It points out some fery interesting text handling features of Logo. In fact Logo is language that is very easy to write adventures in.

The first task in adventure writing is to represent your . information and its relationship to other information. Once we decide on our relationships, they must be presented in ordinary language. With the Logo language, this is very simple. The idea is to let a place name such as FOREST be the name of a listwhose members consist of three lists, one for each of the required types of information.

We define the procedures ITEMS, OPTS, and DESCRIPT, each of which takes a place name for input and returns the corresponding llist of information. The output of ITEMS will be a list of objects, in the case of a place a list of obsticles etc. OPTS could be a list of directions to move such as EAST, WEST eTc. DESCRIPT could be information about the spot where you are.

To play the game type ADVENTURE once it is loaded. At any time you need information about possible direections and/or actions just enter HELP. You can make your requests as descriptive as you like but the name of a direction and/or object must be included. Your object is to bring home the Golden Chalice.....Good Luck!!!

NOTE!!!Indented lines are continuation of previous line.

```
TO ADVENTURE
INIT
PLAY
END

TO INIT
MAKE "DIRS [NORTH EAST SOUTH WEST UP DPOWN ]
MAKE "VERBS [TAKE DROP UNLOCK OPEN ]
MAKE "WORDS [AX KNIFE KEY RING CHALICE GATE ]
MAKEALL BF TEXT "INFO
MAKE "PLACE "FOREST
MAKE "INVENTORY [ ]
CS
DISPLAY [ENJOY YOUR ADVENTURE! ]
DISPLAY [AT ANY TIME YOU CAN TYPE HELP FOR OPTIONS. ]
END
```

```
TO PLAY
DESCRIBE
INTERPRET RL
IF DONE? REPORT STOP
PLAY
END

TO TAKE :OBJ
TEST MEMBER? :OBJ :INVENTORY
IFT DISPLAY SE [ALREADY HAVE ] :OBJ STOP
TEST MEMBER? :OBJ ITEMS :PLACE
IFF DISPLAY SE :OBJ [NOT HERE ] STOP
IF NOT BF :INVENTORY = [ ] THEN DIEPLAY [YOU MUST DROP
    SOMETHING FIRST ] STOP
MAKE "INVENTORY SE :INVENTORY :OBJ
REMOVE :OBJ :PLACE
END

TO DROP :OBJ
TEST MEMBER? :OBJ :INVENTORY
IFF DISPLAY SE [YOU DON'T HAVE ] :OBJ STOP
MAKE "INVENTORY SUBT :OBJ :INVENTORY
PUT :OBJ :PLACE
END

TO UNLOCK :IT
TEST MEMBER? :IT OPTS :PLACE
IFF DISPLAY [NOTHING TO UNLOCK ] STOP
TEST MEMBER? "KEY :INVENTORY
IFF DISPLAY ( SE [CAN'T UNLOCK ] :IT [WITHOUT A KEY] ) STOP
REMOVE "LOCKED :IT
DISPLAY ( SE [THE ] :IT [IS UNLOCKED ] )
END

TO OPEN :IT
TEST MEMBER? :IT OPTS :PLACE
IFF DISPLAY [NOTHING TO OPEN ] STOP
TEST MEMBER? "LOCKED ITEMS :IT
IFT DISPLAY ( SE [THE ] :IT [IS LOCKED ] ) STOP
REMOVE "CLOSED :IT
DISPLAY ( SE [THE ] :IT [IS OPEN ] )
END

TO MAKEALL :LS
IF :LS = [ ] THEN STOP
MAKE FIRST FIRST :LLS BF FIRST :LS
MAKEALL BF :LS
END

TO HELP
DISPLAY [YOU CAN GO ]
DISPLAY :DIRS
DISPLAY [OR ]
DISPLAY :VERBS
PRINT [AN OBJECT. ]
END
```

```
TO INTERPRET :L
IF :L = [HELP ] THEN HELP INTERPRET RL STOP
TEST MEMBER? LAST :L :DIRS
IFT GODIR LAST :L STOP
TEST BOTH MEMBER? FIRST :L :VERBS MEMBER? LAST :L :WORDS
IFT TAKEACTION FIRST :L LAST :L STOP
PRINT [CAN'T UNDERSTAND ]
INTERPRET RL
END

TO INFO
FOREST [AX KNIFE ] [BRIDGE CLEARING CLIFF CLIFF [ ] [ ] ] [IN A
   FOREST ]
BRIDGE [ ] [COURTYARD MOAT FOREST MOAT ] [ON A BRIDGE ]
CLIFF [ ] [ ] [FALLING OFF A CLIFF! ]
CLEARING [KEY ] [[ ] [ ] HOME FOREST ] [IN A CLEARING ]
HOME [ ] [ ] [HOME FROM YOUR JOURNEY ]
MOAT [ ] [ ] [IN A DEEP MOAT! ]
COURTYARD [ ] [GATE [ ] BRIDGE] [IN A COURTYARD BEFORE A
   MAGNIFICENT CASTLE ]
GATE [LOCKED CLOSED ] [HALL [ ] COURTYARD ] [GOING THROUGH AN
   IRON GATE ]
HALL [RING ] [[ ] [ ] GATE [ ] STAIRS ] [IN A GREAT HALL ]
STAIRS [ ] [[ ] [ ] [ ] [ ] ROOM HALL ] [ON MARBLE STAIRS ]
ROOM [CHALICE ] [[ ] [ ] [ ] [ ] [ ] STAIRS ]F [IN AN EXQUISITE
   ROOM ]
KNIFE [A SHARP KNIFE ]
AX [A TRUSTY AX ]
KEY [A RUSTY KEY ]
RING [A GOLD RING ]
CHALICE [A GOLD CHALICE ]
END

TO DONE?
OUTPUT MEMBER? :PLACE [HOME CLIFF MOAT ]
END

TO DESCRIBE
CS
DISPLAY SE [YOU ARE ] DESCRIPT :PLACE
PRINT [ ]
TEST :INVENTORY = [ ]
IFF PRINT [YOU HAVE ] DESOBJS :INVENTORY
TEST ITEMS :PLACE = [ ]
IFF PRINT [YOU SEE ] DESOBJS ITEMS :PLACE
IF NOT DONE? THEN PRINT [THERE IS A ]
DESOPTS :DIRS OPTS :PLACE
END

TO REPORT
DESCRIBE
TEST MEMBER? :PLACE [CLIFF MOAT ]
IFT DISPLAY [YOU ARE IN BIG TROUBLE! ] STOP
TEST MEMBER? "CHALICE :INVENTORY
IFT DISPLAY [YOU WILL LIVE A LONG LIFE. ]
IFF DISPLAY [NEXT TIME BRING THE CHALICE! ]
END

TO REMOVE :OBJ :PLACE
MAKE :PLACE FPUT SUBT :OBJ ITEMS :PLACE BF THING :PLACE
END
```

```
TO SUBT :OBJ :SET
IF :SET = [ ] THEN OUTPUT :SET
TEST :OBJ = FIRST :SET
IFT OUTPUT BF :SET
OUTPUT FPUT FIRST :SET SUBT :OBJ BF :SET
END

TO PUT :OBJ :PLACE
MAKE :PLACE FPUT SE ITEMS :PLACE :OBJ BF THING :PLACE
END

TO ITEMS :P
OUTPUT FIRST THING :P
END

TO DISPLAY :L
PRINT [ ]
PRINT :L
WAIT 120
END

TO OPTS :P
OUTPUT FIRST BF THING :P
END

TO GODIR :DIR
MAKE "NEWPLACE MATCH :DIR :DIRS OPTS :PLACE
TEST :NEWPLACE = [ ]
IFT DISPLAY SE [CAN'T GO ] :DIR STOP
TEST MEMBER? :NEWPLACE [GATE STAIRS ]
IFF MAKE "PLACE :NEWPLACE STOP
TEST ITEMS :NEWPLACE = [ ]
IFF DISPLAY FIRST ITEMS :NEWPLACE STOP
DISPLAY SE [YOU ARE ] DESCRIPT :NEWPLACE
MAKE "PLACE :NEWPLACE
GODIR :DIR
END

TO TAKEACTION :VERB :OBJ
MAKE "OBJ SE :OBJ [ ]
RUN SE :VERB [FIRST :OBJ ]
END

TO MEMBER? :X :L
IF :L = [ ] THEN OUTPUT "FLASE
IF :X = FIRST :L THEN OUTPUT "TRUE
OUTPUT MEMBER? :X BF :L
END

TO DESCRIPT :P
OUTPUT LAST THING :P
END
```

```
TO DESOBJS :L
IF :L = [ ] THEN PRINT [ ] STOP
PRINT FIRST THING FIRST :L
DESOBJS BF :L
END

TO DESOPTS :DIRS :PLACES
IF :DIRS = [ ] THEN PRINT [ ] STOP
MAKE "DIR FIRST :DIRS
TEST FIRST :PLACES = [ ]
IFT DESOPTS BF :DIRS BF :PLACES STOP
TEST MEMBER? :DIR [UP DOWN ]
IFT PRINT ( SE FIRST :PLACES [GOING ] :DIR )
IFF PRINT ( SE FIRST :PLACES [TO THE ] :DIR )
DESOPTS BF :DIRS BF :PLACES
END

TO MATCH :X :XS :YS
IF :XS = [ ] THEN OUTPUT [ ]
IF :X = FIRST :XS THEN OUTPUT FIRST :YS
OUTPUT MATCH :X BF :XS BF :YS
END
```

As can be seen with this program, an adventure is easy to write in Logo, and once written it is easy to modify. After typing and debugging this game. Try your own variations. I have and the possibilities are limited only to memory space.

INSIDE THE TI-99/4A

By
Rick Felzien

The TI-99/4A computer is a very powerful machine, and there is more inside the console than most documentation shows. Suprisingly, though the TI's memory organization is unlike that of any other personal computer, very little has been published on this subject. Let us explore the TI's internal structure and see how this effects tie performance of the computer itself.

A computer's memory is simply a collection of individual memory locations, or addresses. Each location holds one byte of of information, with a value from 0 to 255. Memory is commonly measured in Kilobytes(1024 bytes=1 Kbyte), thus 8K is actually 8192(8*1024).

At the lowest level, a computer can perform only two simple memory operations. It can Read a memory location to see what data it contains, and it can Write a byte of data to a location. There is a limit to the number of locations a computer can address, determined by the addressing capabilities of its microprocessor chip. The TI's TMS9900 microprocessor can address a maximum of 64K(65,536) locations.

Most computers also have a few locations which are neither RAM(Random Access Memory) nor ROM(Read Only Memory). These are control addresses used to control input/output chips. By addressing these locations, the computer controls graphics, disk storage, and periferal access(via the RS-232 card) etc. You may have heard that the TI is a 16-bit microprocessor rather than an 8-bit unit. In fact the TI actually sees its 64K address space as 32,768 two-byte words, rather than 65,536 bytes. Since the microprocessor can manipulate a word either as an entire 16-bit package or as two 8-bit packages, it combines some of the features of both 8-bit and 16-bit processors.

A Memory Map is a chart of the computer's memory organization, showing the location of RAM and ROM locations. Several questions may arise when you scan the memory map for the TI. Like where is the 16K RAM that is supposed to be available in an unexpanded machine? What is a GPL interpreter and what is GROM? Only 8K of the ROM is apparant, so where is the 26K of ROM? Since only 8K is assigned to the module port, how is it possible for modules to hold up to 36K of program information? Finally, how can all of the periferal ROMs occupy the same 8K space?

To find the missing RAM, we need to review some hardware. In addition to the main processor, the TI has a second microprocessor devoted to graphics. The TMS9918A Video Display Processor(VDP) controls sprites, high resolution displays, and other graphics features, and can address a maximum of 16K of RAM. Since this special RAM area can be accessed by the VDP(hence the term VDP RAM), its not included in the general memory map.

Here is the general memory map.

```
                TI-99/4A General memory map.
FFFF |---------------------------------------------|
     |   High Memory Expansion                     | 8K
     |---------------------------------------------|
     |   High Memory Expansion                     |8K
     |---------------------------------------------|
     |   High Memory Expansion                     | 8K
A000 |---------------------------------------------|
     |   Memory-mapped I/O for speech sunthesizer, |
     |   VDP, GROM, Sound Chip, CPU scratchpad     | 8K
     |   RAM at 8300.                              |
8000 |---------------------------------------------|
     |   8K available at the module port           | 8K
     |---------------------------------------------|
6000 |   ROMs for periforal devices-op to 11       |
     |   periforals such as disk controller,RS-232| 8K
     |   interface, etc.                           |
     |---------------------------------------------|
4000 |   Low 8K portion of expansiion RAM          | 8K
     |---------------------------------------------|
2000 |   console ROM, contains part of operating   | 8K
     |   system, GPL interpreter, part of Basic.   |
0000 |---------------------------------------------|
```

As you may already know, the computer can't understand Basic directly. The english words we use in programming such as PRINT, GOTO, GOSUB, etc. must be interpreted(decoded) for the computer. The Processor and its related chips only understand Machine Code, which is a series of the binary 1's and 0's. The interpreter is in a resident ROM in the computer and all programming other than Machine Language must go through this interpreter. In the TI there is also another interpreter called the Graphics Programming Language(GPL) interpreter. This program interprets programming stored in the Graphics Read Only Memory(GROM). GPL is TI's Main language which they developed. It is a very powerful language and recently a package similar to the Editor Assembler for assembly language was made available for programming in the GPL language.

Much of the documentation on the TI-99/4A was kept a big secret by the people at TI. Any and all real information about the inner workings of the computer, its architecture, and its physical design have been discovered by hobbiests like ourselves who have taken the time to play around and discover things about our machines. Those of you who are new to the machine will possibly discover some new tidbit yourselves. By joining User's Groups and by these groups keeping in touch with the swapping of newsletters, and a few periodicals such as MICROpendium, which is dedicated to the TI, we manage to get the word about the machine disseminated.

Here is an expanded memory map of the TI.

```
|---------------|    |------------------|     |------------------|
|             6K|    |High Expansion 8K |  |-|File Buffers      |
|---------------|    |------------------|  | |------------------|
|             6K|    |High Expansion 8K |  |-|Screen            |
|---------------|    |------------------|  | |------------------|
|Module GROMs 6K|    |High Expansion 8K |  | |Disks up to 3  <-|
|---------------|    |------------------|  | |----------------||
|             6K|<-->|memory map I/O 8K |<-  |Disk controller<-|8K
|---------------|    |------------------|  --|------------------|
|             6K|    |Module Port    8K |  |-|P-code card       |8K
|---------------|    |------------------|  | |------------------|
|             6K|    |Periforal ROMS 8K |<-|-|RS-232 interface  |8K
|---------------|    |------------------|  | |------------------|
|Console GROM 6K|    |Low Expansion  8K |  |-|up to 8 periforal |8K
|---------------|    |------------------|    |------------------|
|             6K|    |Console ROM    8K |
|---------------|    |------------------|
```

A person can obtain additional assembly language memory without expanding the computer with the Mini-Memory module. However the programing capability is limited in comparison to the Editor Assembler package and Expansion System.

Hopefully this article has helped those of you who are unfamiliar with the TI-99/4A and if response is good, we will have a few more such items in this newsletter.

SIXTEEN DIGIT KEYPAD USING THE GRAM KRACKER

By STEPHEN PEACOCK, Jacksonville, FL Aug. 1986
Permission is given to reprint this article in newletters, as long as my name remains intact.

When I learned to type, I never learned the number keys. This has been a problem when redefining characters on the TI-99/4A computer. Because of this, I have always wanted a sixteen digit key pad. With the Gram Kracker I now have one.

The keypad is activated by holding the 'CTRL' key down with the left hand. Your right hand will then work the keypad. If the 'CTRL' key is not pressed the keys under the right hand are not changed.

My keypad uses the following keys:

```
        7 8 9 0 become F E D C
        U I O P become B A 9 8
        J K L ; become 7 6 5 4
        N M < > become 3 2 1 0
```

This modification is made to an operation system in Gram 0. If Gram 0 is selected it is always there to use. I have used it in BASIC, Extended Basic, Editor/Assembler and TI-Writer. When using a Terminal Emulator, restore the normal OpSys, as 'CTRL' keys are used.

To make the changes.

1 Have an Op Sys in Gram 0, and activate it.

2 Select '5' to edit memory.

3 Search gram 179C, in hex. You will see:

```
     9F 8F 8C 9B ** ** ** ** 9E 89 8B 80
     ** ** ** ** B7 95 8A 8D ** ** ** **       (** Do not care)
     ** ** ** 8E ** ** ** ** B0 90 9C **
```

4 With the write protect off, change the display to ASCII and type in the following:

```
     D 9 5 0 * * * * E A 6 1
     * * * * F B 7 2 * * * *       (* Do not care)
     * * * 3 * * * * C 8 4 *
```

5 Restore the write protect and save Gram 0. Your sixteen digit keypad is now ready for use.

With a little experimeting you could change any key to any other one you want. Have fun with your Gram Kracker!

Genie mail address S.PEACOCK