MILLERS GRAPHICS

MARCH 1984
$1.50

# THE SMART PROGRAMMER

Well here we are with the March issue and by the great response we have received from the February issue I guess we are on the right track. In regards to our question on the program length in the newsletter SHORT and LONG are running neck and neck at this point. We will keep tabulating the results and let you know how it came out in the next issue. So far the general consensus seems to be for us to keep on going like the February issue. Once again I would like to thank everyone for the comments, suggestions, tips and questions, they have been great so please keep sending them in.

In the Rumor department things have been a little quiet lately but we may have a few goodies for you next month, if we can get confirmation on them. There seems to be a rumor out there that we bought the rights to produce the Extended Basic module. I'm sorry to say that this is NOT true. Two paragraphs of editorial is enough so lets move on to Q & A's.

## Q&A

We have received a few questions on negative and positive duration CALL SOUND statements so lets see if we can explain them a little better.

When the program encounters a sound statement with a negative duration such as CALL SOUND(-400,660,0) it immediately halts any other sounds that are on and starts up the new sound. When the program encounters a sound statement with a positive duration such as CALL SOUND(400,660,0) it waits until the previous sound has completed its duration time before the new sound starts up. I believe that the program listed below will help to clarify this a little more. When you run this program it will first execute a series of positive duration sounds and then it will execute the same sounds but with a negative duration.

Positive duration sounds are mainly used for songs and sound effects that require each sound to be on for a specific time. Positive duration sounds also make great time delays since the duration is in milliseconds (duration/1000=seconds). The following two sound statements create a 2 second delay before the program continues execution. CALL SOUND(2000,-4,30) :: CALL SOUND(1,-4,30). You can easily add some lines of code in between the two sound statements, such as initializing variables and strings or moving a sprite for a given time period, so that the time is not a total waste. This is what we used for the timing on our shooting routine in the Smart Programming Guide for Sprites.

Negative duration sounds are used for general input prompts or warning sounds as well as for sound effects that require immediate response or rapid changes in frequency and/or volume. The BEEP for INPUT and the HONK for syntax and other errors and warnings the the 99/4A generates are both negative duration type sounds.

```
10 CALL CLEAR :: A$(0)="POSI
TIVE" :: A$(1)="NEGATIVE" ::
 A=1

20 FOR I=1 TO 24 :: DISPLAY
AT(I,11)ERASE ALL:A$(ABS(A<0
)):: CALL SOUND(A*400,1070-I
*40,I,1080-I*40,I+2):: NEXT
I :: A=-A :: GOTO 20
```

Looks like this was the month for Hardware questions so lets dig into them. Before we start I must say that these are my opinions based on our own research.

Can Extended Basic be but on a card for the P-Box?

Yes it can but it would be fairly tricky on the 99/4A. The way the 4A is currently memory mapped leaves very little room in the CPU address space for the 36K this language requires. You can't use the 8K console ROM space since it is already full. You could use the 8K low memory expansion space but then it wouldn't be compatible with any programs that use Assembly language subroutines since they reside here. You could use the 8K DSR space if you wanted to give up use of your disk drives and/or RS232 since they are paged into this space. The 8K cartridge space is open for its use so it could be mapped into this space and with some very fancy bank switching and logic checking, approximately five 8K ROM banks, it might work out OK. Since it would be in ROM it would have to be rewritten in Assembly language and it would need a GPL link routine to access the info in GROM chip 0. 24K of Extended Basic is currently written in GPL language and this language will not run in ROM or RAM without going through some sort of GROM simulator (auto-incrementing addressing).

The other choice would be to put 12K of it in ROM, like it is now, and the rest of it in GROM. There might be a problem with accessing GROM out of the side port though because not all of the cartridge port lines go out the side and TI has stated that they were at one time considering eliminating the ones that currently are there. This may mean that not all 99/4A's have the same GROM lines out the side port, I'm not sure if they ever did this. There wouldn't be any cost savings with it on a card and depending on how the card was set up you may or may not be able to access another module that is plugged into the cartridge slot. Speaking of costs don't forget about the costs for purchasing the rights from TI, that could be quite a chunk of change. I don't currently know of anyone that is planning on producing such a card and I think if someone other than TI buys the rights and manufactures it they will probably follow the current and tested method of putting it in a module.

Can you put more than one memory expansion card in the P-Box?

Yes you can. However they will act in parallel and as such you will still only have 32K of memory expansion. Lets talk about about expanding the memory on our computers for a moment. The TMS 9900 microprocessor and its current memory addressing scheme will only allow 64K of memory addressing. 32K of this is reserved for system operation, 8K console ROM, 8K DSR space, 8K cartridge space and 8K of RAM space for memory mapped devices and the CPU's scratch pad. This leaves us with 32K of address space for memory expansion, that is the maximum expansion RAM that our computer can address without messing up the compatibility of everything else. We are limited to 64K of addressing because the address bus is 16 bits wide. So with every bit on, equal to 1, we have a binary number with 16 ones in it, ie; 1111111111111111. If you convert this into an unsigned, not two's compliment, decimal number it equals 65,535. In two's compliment binary, signed numbers, this equals -1. Now 65,535 plus address 0 equals 65,536 possible address locations. 1K is equal to 1,024 bytes or address locations so 65,536 divided by 1,024 equals 64K.

There are other 16 bit machines out there that allow memory expansion beyond 64K such as the IBM PC and TI PC but their microprocessor still only talks to 64K blocks of memory at a time even if they have a 512K RAM card installed. Many of these machines have added some sort of fancy "Shadowing" , DMA (Direct memory access) or Segmented Register technique to allow the microprocessor to easily page in different 64K sections of memory as they are needed. Our 4A's do not have this so 64K is max and 32K of directly accessible Expansion RAM is max without some special hardware AND special software on the expansion card. I know that there are some 128K expansion cards out there but remember they only bank switch 32K of it into the address space at one time. The rest of it can be used as print spoolers, buffers, or a RAM disk but it requires special software to make it work. ( I think I stuck my neck out on that one, I can only imagine the letters that will be coming in). Lets move on to a Firmware question about the Editor/Assembler cartridge.

First off let me say that this module seems to be getting hard to find. If you can find one and you have or are planning on getting Memory Expansion and a disk drive I strongly recommend that you pick it up! I believe that in the U.S. it has a suggested retail price of 39.95 and for that amount its one heck of a bargain even if you don't plan on writing Assembly language programs.

This module comes with the Editor / Assembler manual, two floppy diskettes and the module itself. If you are not planning on writing Assembly language and you are not interested in the reference info in the appendices, then this book and a cup of hot chocolate is probably the best cure for insomnia you'll ever find. However if you are interested in Assembly language then, even though the book is boring, you need it!!! It is not, however, an introduction to Assembly it is only a reference manual of Assembly language opcodes, syntaxes.

One of the two diskettes contains the assembled, object code, game of Tombstone City as well as the documented source code, what the programmer typed in to get the object code after it was run through the Assembler. The other diskette contains the Assembler, The Editor, a debugger (both the source and object code) and some utilities. So what is in the module? Not much! The module has the input prompts for using it, it loads some utilities into low memory expansion, such as the LOADER, VDP read and write routines, DSR link, GPL link etc. It also sets up the VDP memory map for its use of VDP RAM and it adds INIT, LINK, PEEK, LOAD, PEEKV, POKEV and CHARPAT CALL's to console Basic. There is 6K of GROM and no ROM in this module, most of the goodies are on disk.

The Editor that loads and runs with this module is one of the best windowed 80 column under $40 word processors you can buy. It will do FIND, REPLACE, MOVE, INSERT, COPY, SHOW, DELETE and it has user setable Tabs. It will not right justify your type but it is very easy to use, what you see is what you get. The files can be saved to disk as either DISPLAY FIXED 80 or DISPLAY VARIABLE 80 and they can be easily OPENed and written to or read from in Basic and Extended Basic. The DIS/VAR 80 files are SEQUENTIAL and the DIS/FIX 80 files are RELATIVE type files.

With interaction like this between Extended Basic and Assembly language source code or word processed text and or DATA there are quite a few interesting possibilities. How about an Extended Basic program that writes Assembly source code. Hmmm.. sounds like the start of a two pass compiler!! Anybody have any spare time? How about an Extended Basic Mail-merge program for form letters? What if we go the other way and use it as a full screen word processing type editor for Extended Basic programs. Then we could use another Extended Basic program to read it in and convert it from ASCII into a MERGEable token type file which could then be loaded with MERGE DSK1.xxxxxxx . (See the program that converts MERGE token files into ASCII on page 11). Of course all of this is also possible, except for the actual assembly conversion, with TI-Writer. Sounds like fun, I just wish I had a little more spare time to play with it. If someone will write it we will publish it. Well enough of that for now.

Before I wrap up this article I must add that one of the other main reasons for picking up the Editor / Assembler module is so you can load and run TI Forth and TI Forth programs! If you haven't seen what the 99/4A can do in Forth run down to the next users group meeting in your area and ask for a demo, I think you will be impressed.

One last minute tip on using the Editor / Assembler module which just came to us from Danny Michael of Florence, Alabama. To get the LIST FILE sent out through the parallel port you need to type in PIO. or PIO.EC in response to the list file prompt. It needs at least a period after PIO for it to accept this as a valid I/O option. Danny also sent us some other interesting items which we will be publishing in the next issue. Thanks Danny.

---

How do you get more than an 80 column output on the printer in either compressed mode or with a 15" carriage?

The TI RS232 card automatically sends a carriage return character, CHR$(13), after every 80 characters. To defeat this you will need to OPEN the device with the carriage return off, this will also turn off the line feeds. The statement that does this would look like this for parallel output. OPEN #1:"PIO.CR" . The only problem with using this is that you will now have to send your own carriage return and line feed commands to the printer at the end of the string. To do this you need to include CHR$(13)&CHR$(10) in the print statements that are sent out to the printer. If you are using the full width of the carriage you only need to send them at the end of each string such as:

```
PRINT #1:A$&CHR$(13)&CHR$(10)
```

However if you are not using the full width of your carriage (ie: 15 inch carriage with 8 inch paper) then you will need to send CHR$(13)&CHR$(10) at the appropriate column to keep the print head from going off the edge of the paper.

---

Can TI Forth subprograms be used with Extended Basic.

No. TI Forth maps out the Expansion Memory, VDP RAM and Scratch Pad RAM completely different than Extended Basic does. It uses the entire area of scratch pad RAM that the Extended Basic interpreter normally uses so the two would confuse each other. Also the area in high Mem-Expansion that is normally used to store an Extended Basic program is used by Forth as its stack area and terminal input buffer. All Forth programs must have the Forth Interpreter, 99/4A Forth support and resident Forth vocabulary loaded into memory in order to run.

All languages need some form of interpreter and vocabulary some where in memory. The difference between Basic or Extended Basic and Forth is that these items are in ROM and GROM where Forth is loaded into RAM on the 99/4A. Many of the PC type computers out there have very little ROM and all of their languages are loaded into RAM so they are really dumb computers until something is loaded into memory.

---

We received the following information on Signalman Mark III, Mark VII and Volksmodem modems from a dealer by the name of Tom Knight. He can be reached at 7266 Bunion Dr., Jacksonville, FL 32222. His phone number is (904) 778-4507. The following is an excerpt from his informative letter on these modems.

"The main reason that I am writing is your reference to the MK-III modem in the current news letter. Are you aware that Signalman makes another modem for the TI? It is the Volksmodem and in my opinion is better than the MK-III. Mainly because it is a true direct connect modem, you can use it with any type of telephone. With the MK-III you must have a certain type phone. It also comes with a Source subscription, as do all of their modems. Their MK-VII modem will also work but requires a slight modification. (Pins 1 and 2 must be swapped). Both the Volksmodem and the MK-VII connect directly into the wall and the phone is plugged into them. With the MK-III you put it between the base and the hand set. The suggested retail price for the Volksmodem is about $93.00 with cables. Several members of our Users Group are using these modems with no problem. Also the MK-VII has auto originate and auto answer capability as does the Volksmodem."

Thanks for the info Tom.

---

## *OOPS!*

Well we did it again. Looks like we had a few more goofs in the February issue. First off I'd like to apologize to CorComp Inc., I should have checked out the RS232 card for myself. There is nothing wrong with the serial pin outs numbered 2 & 3 they are the same as the TI card, sorry about the mistake and any troubles this might have caused anyone or any of the CorComp dealers and distributors.

In the VDP RAM map on page 9 I stated that the crunch buffer was included with the other items that are between >0370 and >077F. That was wrong. The crunch buffer, which we will explain when we map out VDP RAM, is located at >0820 - >08BF.

Also on page 9, at the top of the right hand column there is a small mistake. The top of this column should read:

**CONSOLE GROM >0000 - >57FF 18K Bytes**

That about does it for the OOPS! column for this month. At least its getting shorter.

The New
## EXPANSION SYSTEM

The latest news from CorComp Inc. is their brand new EXPANSION SYSTEM for your TI 99/4A computer.

The EXPANSION SYSTEM has provisions for installing 2 slimline disk drives or 1 full height drive and it contains the powerful SYSTEM motherboard. The following items that expand the power and versatility of your 99/4A computer are included on the motherboard.

1. 32K of Expansion RAM Memory which will allow you to run TI Writer, TI LOGO or LOGO II, TI Multiplan, TI Forth and Assembly language programs as well as larger and more powerful Extended Basic programs.

2. A Double Sided - Double Density disk controller. This controller will control up to 4 disk drives and they can be accessed as either DSK or dsk (upper or lower case). With 2 slimline Double Sided Double Density disk drives mounted in the box you will have 720K of on line rapid access disk storage. With four drives hooked up you will have 1.4 megabytes of on line storage. The tests we saw indicated that this new disk controller loads programs 2 - 4 times faster than the TI disk controller. The DSR ROM will contain the necessary support for loading "Load and Run" type Assembly language programs without the Editor / Assembler command module. This same ROM will also add CALL POKEV and CALL PEEKV for accessing VDP RAM and they can be used as commands or program statements.

3. An RS232 interface with 2 serial outputs and 1 parallel output. The serial outputs are TI compatible and the parallel is a true Centronics output. This allows you to hook up printers, plotters and modems to your computer.

4. Specially designed power supply that will power 2 slimline disk drives, and the SYSTEM motherboard.

5. The FLEX cable interface for hooking all this power up to your computer. The cable is a small flexible round one (not a fire hose) that plugs into the side of your computer via a small L type connector. This connector directs the cable towards the back of the computer instead of straight out to the side.

---

The EXPANSION SYSTEM, which is about half the size of TI's box, has some illuminated graphics and the power switch on the front panel. The graphics return information on the Flex cable interface, 32K RAM, RS232, Disk controller, disk Side 1, disk Side 2, the disk Index mark and the Option slots.

The disk controller will control just about any 5 1/4 inch disk drive from full height single sided - single density to double sided - double density slimlines. There are provisions in the disk controller section of the motherboard to set up the head seeking time (track to track) to match some of the faster (more expensive) disk drives out there. The disk manager program is supplied on disk with the box. This is a completely new program with some very nice enhancements added to it! At the time this article was written there were other possible future options being tested for the Expansion System. These items weren't fully tested yet so I'll let you know about them next month.

The EXPANSION SYSTEM will be available through your local dealer starting at the end of March. PLEASE contact your local dealer for additional information and pricing.

CorComp is also currently working on a stand alone RS232 interface for the 99/4A. This RS232 unit will plug directly into the side of your computer and it will be shaped to match the 99/4A profile. It will have a serial port and a parallel port. This will allow you to hook up a modem, and/or a printer or plotter to your computer. The suggest retail price is unknown at this time. The other item they are currently working on is a Disk Controller card for the TI Peripheral Expansion box. This controller card will have all of the same features as the controller that is on the motherboard in the Expansion System. It will also come with the same Disk Manager program on diskette and it should be available around the same time as the Expansion System. The price is unknown at this time.

## *PEEKING AROUND*

This month we will look at Extended Basic's use of Low and High Memory Expansion (the Editor/Assembler and Basic use is different). If you do not execute CALL INIT then Extended Basic does not use Low Memory Expansion. However when you use CALL INIT X-Basic moves a large amount of data into this space. The data is mainly used by Assembly language programs and subprograms. Some of this data also directs the loader, CALL LOAD("DSK1.xxxx"), where in memory to load the assembly program.

**XML Link to name link routine:**
In Extended Basic the starting address of this routine is pointed to by >2000. This routine is used to find a specific name in the REF/DEF table. First it compares the value at >2004, LFA, to >4000, if >2004 contains >4000 then it jumps out of this routine and returns a name NOT FOUND error. If the value at >2004 is less than >4000, ie >3FF8, it then compares the called name, ie; CALL LINK("**DEBUG**"), which is stored starting at FAC, >834A, against the characters starting at LFA. It compares them two at a time, 16 bits, until all of the characters have been tested or until one of the characters don't match. If they all match it then loads the starting address of the routine into R0 and branches to it. If they don't match it adds 8 to R1, which is the pointer to the start of the name, and starts the match all over again. When R1 equals >4000 it leaves this routine and returns a NOT FOUND error.

**FFA First Free Address in low mem-exp:**
The value at >2002 points to the first unused byte in low memory expansion. This value is used by the loader to determine where to place the assembly program that is being loaded. After the program is loaded this value is updated by the loader to point the new FFA. NOTE: This updating does NOT occur on AORG type programs!!!

**LFA Last Free Address in low mem-exp:**
The value at >2004 points to the start of the REF/DEF table or, if there isn't a REF/DEF table, it points to the end of low mem-expansion at >4000. As items are added to the REF/DEF table by the loader this value is updated by subtracting 8 bytes for every entry.

**CALL INIT constant:**
The value at >2006 is a specific value of >AA55 that some X-Basic subprograms use to check to see if CALL INIT has been executed. When your program executes CALL LINK or CALL LOAD they check to see if the value at >2006 is equal to >AA55. If it is not, it issues an error message, if it is, it allows the routine to be executed. Because these routines check for this value in low mem-expansion you can not execute CALL LOAD or CALL LINK without it.

**UTILITY VECTOR TABLE:**
A vector table, of which there are many throughout ROM, RAM and GROM, contains pairs of pointers for various items. In this vector table the pointers indicate where the utility routine's workspace will be and where the utility routine starts. When you are writing an Assembly language program that is loaded through X-Basic you need to set up the EQUATES to point into this table. For detailed info on these utilities and how they are used see chapter 17 in the Editor/ Assembler manual. The following routines are vectored from this table:

**NUMASG**
This routine places (assigns) a value(s) into a numeric variable(s) that is used in a CALL LINK statement. In other words, it places the value that is in the Floating point ACcumulator, FAC =>834A, into the Numeric Value table in high mem-expansion and sets up the pointer in the Symbol table in VDP RAM to point to it. This allows your X-Basic program to use a value(s) that your Assembly program generates.

**NUMREF**
This is the opposite of NUMASG. This utility allows your Assembly program to use values that are passed to it from your X-Basic program via CALL LINK. In other words, it looks up the variable name in the Symbol table, finds out where it is in the Numeric Value table and moves it into FAC for your Assembly program's use.

**STRASG & STRREF**
These act like NUMASG & NUMREF except they work on string variables. In STRASG strings are passed to VDP RAM from mem-expansion. In STRREF they are passed the other way. The starting address of the string in mem-expansion is in register 2 of the workspace.

**XMLLNK**

This utility allows your Assembly program to link to a routine in ROM or to a routine in mem-expansion. These routines use the GPL workspace at >83E0, the Utility workspace at >2038 is only used to store the return address. For more info on this routine see chapter 16 in the Editor / Assembler manual. When we start mapping out ROM we will discuss the various ROM routines and their locations.

**KSCAN**

This sets up the workspace, GPL, and return address and then branches, BL, to >000E which is the actual keyboard scanning routine. After the scan is complete it returns to this routine which then restores the utility workspace and return address and then returns, RTWP. Here is the entire KSCAN routine, this does not include the SCAN routine in ROM.

```
KSCAN  LWPI >83E0          GPL workspace
       MOV R11,@>2038+22   Save GPL rtn add
       BL @>000E           SCAN routine ROM
       LWPI >2038          Utility workspace
       MOV R11,@>83E0+22   Restore rtn add
       RTWP                return
```

**VSBW, VMBW, VSBR, VMBR & VWTR**

The first four routines READ and WRITE single or multiple bytes to and from VDP RAM. The last routine, VWTR, writes a byte to a VDP register. We will discuss the VDP registers and their use when we start mapping out VDP RAM

VSBW writes a single byte from the workspace to VDP RAM.
VMBW writes multiple bytes from CPU RAM via the workspace to VDP RAM.
VSBR reads a single byte from VDP RAM into the workspace.
VMBR reads multiple bytes from VDP RAM into CPU RAM via the workspace.
VWTR writes a single byte from the workspace to a VDP register.

The following is a complete disassembly of these routines with the EQUATES. The workspace for these routines is located at >2038.

```
VDPRD   EQU >8800   VDP read  data address
VDPWD   EQU >8C00   VDP write data address
VDPWA   EQU >8C02   VDP read/write address
R2LB    EQU >2038+5 (>203D) R2's Lower byte
```

```
VSBW    BL   @SETWDA  set up write out addrs
        MOVB @2(R13),@VDPWD   write out byte
        RTWP                          return

VMBW    BL   @SETWDA  set up write out addrs

VWTLOP  MOVB *R1+,@>VDPWD     write out byte
        DEC  R2          decrement byte count
        JNE  VWTLOP     if more to write jump
        RTWP               else        return

VSBR    BL   @SETRDA  set up read from addrs
        MOVB @VDPRD,@2(R13)      read a byte
        RTWP                          return

VMBR    BL   @SETRDA  set up read from addrs

VRDLOP  MOVB @VDPRD,*R1+         read a byte
        DEC  R2          decrement byte count
        JNE  VRDLOP      if more to read jump
        RTWP                else       return

VWTR    MOV  *R13,R1      get reg # and value
        MOVB @1(R13),@>VDPWA write out value
        ORI  R1,>8000    set for write to reg
        MOVB R1,@VDPWA        write out reg #
        RTWP                          return

SETWDA  LI   R1,>4000     set to write to VDP
        JMP  WVADD

SETRDA  CLR  R1          set to read from VDP

WVADD   MOV  *R13,R2          get VDP address
        MOVB @R2LB,@VDPWA      write low byte
        SOC  R1,R2           adjust write bit
        MOVB R2,@VDPWA        write high byte
        MOV  @2(R13),R1     get CPU RAM addrs
        MOV  @4(R13),R2        get byte count
        RT                            return
```

NOTE: These utilities use the Utility workspace at >2038 which is in the slower 8 bit RAM. If you write your own utilities try and place your workspace in the 16 bit RAM scratch pad area for faster execution time.

**ERR**

The ERR utility transfers control to the basic error reporting routine in GROM. This allows your Assembly language routine to use the standard error messages as called out in the Editor / Assembler manual on page 288. This routine, after setting up the workspace, branches to routines in console ROM which then branches to the error routine in GROM and then returns controll back to your basic program.

## Extended Basic

### LOW MEMORY EXPANSION after CALL INIT

| Address | Value | Description of address |
|---|---|---|
| >2000 | >205A | XML link to name link routine pointer. |
| >2002 | >24FA | First Free address in low mem-exp. |
| >2004 | >4000 | Last Free address in low mem-exp. |
| >2006 | >AA55 | Constant that indicates CALL INIT has been executed. |
| >2008 | **UTILITY VECTOR TABLE** (ie: BLWP @KSCAN ) | |
| >2008 | >2038 | Utility workspace pointer for BLWP @NUMASG |
| >200A | >2096 | **NUMASG** Utility starting address. |
| >200C | >2038 | Utility workspace pointer for BLWP @NUMREF |
| >200E | >217E | **NUMREF** Utility starting address. |
| >2010 | >2038 | Utility workspace pointer for BLWP @STRASG |
| >2012 | >21E2 | **STRASG** Utility starting address. |
| >2014 | >2038 | Utility workspace pointer for BLWP @STRREF |
| >2016 | >234C | **STRREF** Utility starting address. |
| >2018 | >2038 | Utility workspace pointer for BLWP @XMLLNK |
| >201A | >2432 | **XMLLNK** Utility starting address. |
| >201C | >2038 | Utility workspace pointer for BLWP @KSCAN |
| >201E | >246E | **KSCAN** Utility starting address. |
| >2020 | >2038 | Utility workspace pointer for BLWP @VSBW |
| >2022 | >2484 | **VSBW** Utility starting address. |
| >2024 | >2038 | Utility workspace pointer for BLWP @VMBW |
| >2026 | >2490 | **VMBW** Utility starting address. |
| >2028 | >2038 | Utility workspace pointer for BLWP @VSBR |
| >202A | >249E | **VSBR** Utility starting address. |
| >202C | >2038 | Utility workspace pointer for BLWP @VMBR |
| >202E | >24AA | **VMBR** Utility starting address. |
| >2030 | >2038 | Utility workspace pointer for BLWP @VWTR |
| >2032 | >24B8 | **VWTR** Utility starting address. |
| >2034 | >2038 | Utility workspace pointer for BLWP @ERR |
| >2036 | >2090 | **ERR** Utility starting address. |
| >2038 | **UTILITY WORK SPACE STARTS HERE** | |
| | R0-R15 | |
| | | |
| >2057 | End of work space | |
| >2058 | | |
| >205A | Start of XML link to name link routine. | |
| | (Finds the name in the REF/DEF Table) | |
| >2090 | Start of ERR Routine. | (Return Error code to basic) |
| >2096 | Start of NUMASG Routine. | (Numeric Assignment) |
| >217E | Start of NUMREF Routine. | (Numeric Reference) |
| >21E2 | Start of STRASG Routine. | (String Assignment) |
| >234C | Start of STRREF Routine. | (String Reference) |
| >2432 | Start of XMLLNK Routine. | (Link to system Utilities) |
| >246E | Start of KSCAN Routine. | (Keyboard Scan) |
| >2484 | Start of VSBW Routine. | (VDP single byte write) |
| >2490 | Start of VMBW Routine. | (VDP multiple byte write) |
| >249E | Start of VSBR Routine. | (VDP single byte read) |
| >24AA | Start of VMBR Routine. | (VDP multiple byte read) |
| >24B8 | Start of VWTR Routine. | (Write to VDP register) |
| | (NOTE: No GPLLNK or DSRLNK in X-Basic CALL INIT) | |
| | | |
| >24FA | First Free Address in Low Mem-Exp. pointed to by >2002 | |
| | | |

```
|        | *-------------------------------------------------------*  |
|        | *        The REF/DEF Table resides at the end of         *  |
|        | *     Low Memory Expansion. Each entry is 8 bytes long.   *  |
|        | *     6 for the Name and 2 for the starting address.      *  |
|        | *     CALL INIT in X-Basic leaves this space empty.       *  |
|        | *-------------------------------------------------------*  |
|  >3FF0 | DEF Name (CALL LINK or BLWP @) 6 characters.               |
|  >3FF6 | Start address of the above routine, 2 bytes.               |
|  >3FF8 | DEF Name (CALL LINK or BLWP @) 6 characters.               |
|  >3FFE | Start address of the above routine, 2 bytes.               |
|>3FFF   |END OF LOW MEMORY EXPANSION                                 |
+---------------------------------------------------------------------+
```

**Extended Basic**

**HIGH MEMORY EXPANSION usage**

```
+---------------------------------------------------------------------+
|>A000   |START OF HIGH MEM-EXPANSION                                  |
|        |      (If Mem-Exp is present then the value at >8389         |
|        |       will be >E7 while the program is running)             |
|        |                                                             |
|        |    ------------------------------------------------------   |
|        |     NUMERIC VALUE TABLE  (in RADIX 100 notation)            |
|        |                                                             |
|        |     Starting point of the Symbol table in VDP RAM is        |
|        |     pointed to by >833E while the program is running.       |
|        |     The Symbol table then points into the Numeric value     |
|        |     table for each of the variable names.                   |
|        |    ------------------------------------------------------   |
|        | Highest Free Address in Mem-Exp. pointed to by >8386        |
|        |    ------------------------------------------------------   |
|        |     LINE NUMBER TABLE - 4 Bytes per entry.                  |
|        |                                                             |
|        |     | Line # = 2 Bytes | Start Address of line = 2 bytes |  |
|        |     Line numbers are always stored highest # to lowest #    |
|        |     Starting address of this table is pointed to by >8330   |
|        |     Ending   address of this table is pointed to by >8332   |
|        |     Current line number being referenced                    |
|        |                      in this table is pointed to by >832E   |
|        |    ------------------------------------------------------   |
|        |     PROGRAM SPACE (Last line entered is at the top)         |
|        |                                                             |
|        |     Start of program space = (value at >8332)+1             |
|        |     Programs reserved words have been converted to Token values |
|        |     and the line numbers are removed from the beginning of  |
|        |     each line. The format for each line is as follows:      |
|        |                                                             |
|        |           1st Byte = Number of bytes for the line           |
|        |    Following Bytes = (Start Address) Actual line code with  |
|        |                      Token values replacing reserved words. |
|        |          Last byte = >00                                    |
|        |                                                             |
|  >FFE7 |  Highest address to be used in Mem-Exp. pointed to by >8384 |
|        |                                                             |
|  >FFFC |  Workspace for LOAD Function.        (non-maskable interrupt, |
|  >FFFE |  Address for start of LOAD Function.         not DSK1.LOAD) |
|>FFFF   |END OF HIGH MEMORY EXPANSION                                 |
+---------------------------------------------------------------------+
```

## LINE SIZE

This program PEEKs into high memory expansion and displays information about your program. After you type it in save it out to disk as a MERGE type file. Then you can MERGE it back onto the top of anyone of your programs and run it. This program displays the bytes per line, where the line number table is, where each line starts in memory and what the total size of your program is. You can display the results on the screen or send them out to your printer. You may need to modify the OPEN statement on line 1 to match your printer. On line 4 we are adding 5 bytes to the D value, 4 to compensate for the line number table entries and 1 to compensate for the line length indicator. While this program is running you can stop and restart the scrolling by pressing any key.

```
1 CALL CLEAR :: PRINT :"Outp
ut to Printer? (Y/N) N" :: A
CCEPT AT(23,26)SIZE(-1)VALID
ATE("YN"):A$ :: IF A$="Y" TH
EN OPEN #1:"PIO" :: P=1

2 CALL CLEAR :: CALL PEEK(-3
1952,A,B,C,D):: A=A*256+B-65
536 :: C=C*256+D-65536 :: PR
INT #P:"    PROGRAM INFORMAT
ION": :"Line Number Table"

3 PRINT #P: :"Start Address
";A:"End   Address ";C: : :"
 Line    Bytes   Start":"  N
umber   Used    Address":" --
----  -----   -------"

4 FOR I=C-3 TO A STEP -4 ::
CALL PEEK(I,B,D,E,F):: B=B*2
56+D :: E=E*256+F-65536 :: C
ALL PEEK(E-1,D):: D=D+5

5 PRINT #P,USING "  #####
 ###   ######":B,D,E :: T=T+D
 :: CALL KEY(0,D,E):: IF E T
HEN CALL SCREEN(6):: GOSUB 7

6 NEXT I :: A=(A-C-1)/-4 ::
PRINT #P: : :TAB(8);"Total B
ytes =";T:"   Number of Line
s =";A:"Average Bytes/Line =
";INT(T/A):: STOP

7 CALL KEY(0,D,E):: IF E<1 T
HEN 7 ELSE CALL SCREEN(8)::
RETURN
```

## PEEKer

This program will allow you to PEEK anywhere in CPU memory, >0 - >FFFF, and it will display both the numeric value and the ASCII characters for the values on the screen. You may input your starting and ending address as either HEX or Decimal values and it will display the values in these memory locations in the same number base. You may want to combine LINE SIZE and PEEKer together so you can easily find and display your program as it sits in memory.

```
100 ON WARNING NEXT :: CALL
CLEAR :: H$="0123456789ABCDE
F" :: PRINT "DEPRESS YOUR AL
PHA LOCK KEY": :"PRESS LETTE
R FOR INPUT BASE": :

110 PRINT : :"D =DEC #   H =
HEX #": : :: CALL SOUND(40,6
60,9)

120 CALL KEY(0,K,S):: IF S<1
 THEN 120 ELSE ON POS("DH",C
HR$(K),1)+1 GOTO 110,140,180

130 INPUT A$&" DEC ADDRESS=
":EA :: IF EA<-32768 OR EA>6
5535 THEN 130 ELSE EA=INT(EA
+65536*(EA>32767)):: PRINT :
: RETURN

140 A$="START" :: GOSUB 130
:: SA=EA :: A$="END  " :: GO
SUB 130 :: IF EA<SA THEN 140

150 FOR I=SA TO EA STEP 8 ::
 CALL PEEK(I,A,B,C,D,E,F,G,H
):: PRINT USING "###### ###
### ### ### "&CHR$(A)&CHR$(B
)&CHR$(C)&CHR$(D):I,A,B,C,D

160 PRINT USING "###### ###
### ### ### "&CHR$(E)&CHR$(F
)&CHR$(G)&CHR$(H):I+4,E,F,G,
H :: NEXT I :: GOTO 110

170 PRINT A$&" HEX ADDRESS="
: : :: ACCEPT AT(22,20)BEEP
SIZE(4)VALIDATE(H$):EA$ :: R
ETURN

180 A$="START" :: GOSUB 170
:: SA$=EA$ :: A$="END  " ::
GOSUB 170 :: IF EA$<SA$ THEN
 180 ELSE GOSUB 210 :: SA=T
:: SA$=EA$ :: GOSUB 210
```

```
190 EA=T :: FOR I=SA TO EA S
TEP 6 :: A=I :: GOSUB 220 ::
 A$=HEX$&" " :: FOR T=I TO I
+5 STEP 2 :: CALL PEEK(T,A,B
):: B$=B$&CHR$(A)&CHR$(B)

200 A=A*256+B :: GOSUB 220 :
: A$=A$&HEX$&" " :: NEXT T :
: PRINT A$&" "&B$ :: B$="" :
: NEXT I :: GOTO 110

210 T=0 :: SA$=SEG$("0000",1
,4-LEN(SA$))&SA$ :: FOR I=1
TO 4 :: T=T+(POS(H$,SEG$(SA$
,I,1),1)-1)*16^(4-I):: NEXT
I :: T=T+65536*(T>32767):: R
ETURN

220 A=A+65536*(A>32767)

230 HEX$=SEG$(H$,(INT(A/4096
)AND 15)+1,1)&SEG$(H$,(INT(A
/256)AND 15)+1,1)&SEG$(H$,(I
NT(A/16)AND 15)+1,1)&SEG$(H$
,(A AND 15)+1,1):: RETURN
```

## MERGE / READ

This program was sent to us from Barry Traver. When you run this program it will ask you for the file name of a program that was saved on DSK1 with the MERGE option. It will then OPEN this file and read it in. As it reads it in it converts the Token values into their actual reserved words and displays them on the screen. By playing around with this and becoming familiar with the token values you could easily reverse this action. Then you could have your program OPEN a file and PRINT #1 out to it, this means your program could write actual program lines. These lines could then be MERGEd back in and run. You could use it to write CALL SOUND(xxx,xxx,x) lines for music or sound effects or CALL CHAR lines for characters. This could open up a whole new aspect of programming.

Thanks Barry.

```
10 ! MERGE/READ BY
        B.A. TRAVER
        552 SEVILLE ST.
       PHILA., PA 19128
 (FOR FURTHER INFORMATION,
       SEND S.A.S.E.)

20 !  THIS PROGRAM WAS
  INSPIRED BY TWO PROGRAMS
       BY JOHN CLULOW.

30 ! THE PURPOSE OF THE
PROGRAM IS TO ASSIST IN THE
READING AND INTERPRETATION
OF PROGRAMS STORED IN MERGE
FORMAT ON DISK.

40 DATA ELSE

50 CALL CLEAR :: DIM C$(254)
,Y(200):: GOTO 60 :: C,CL,I,
J,K,L,L1,L2,R,S,S1,S2,N$,X$
:: CALL KEY :: !@P-

60 PRINT "MERGE/READ PROGRAM
": :"BY BARRY A. TRAVER": :
:"WANT COMMENTS (Y/N)? ";

70 CALL KEY(0,K,S):: IF S=0
THEN 70 ELSE PRINT CHR$(K);:
: IF K=78 THEN 170 ELSE IF K
<>89 THEN 70

80 CALL CLEAR :: PRINT "  SO
ME COMMENTS ON PROGRAMS IN M
ERGE FORMAT:": : :"  EACH PR
OGRAM LINE IS REP-":"RESENTE
D BY A STRING OF"

90 PRINT "CHARACTERS.": :"
THE FIRST TWO CHARACTERS  RE
PRESENT THE LINE NUMBER;  TH
E LAST CHARACTER IS ALWAYSCH
R$(0)."

100 PRINT :"  MANY ASCII CHA
RACTERS HAVESPECIAL MEANINGS
, E.G.,      CHR$(154) REPRES
ENTS ""REM""  AND CHR$(157)
REPRESENTS     ""CALL"".": :
: :

110 GOSUB 370

120 PRINT "  WITH INSPECTION
  OTHER    PATTERNS CAN BE D
ISCOVERED.": :"  FOR EXAMPLE
, CHR$(200)    INDICATES THA
T THE NEXT"

130 PRINT "CHARACTER WILL IN
DICATE THE LENGTH OF THE STR
ING TO FOL-":"LOW (IF YOU CA
N FOLLOW":"THAT!)."

140 PRINT :"  ""LINE 65535""
 OR CHR$(255)&":"CHR$(255) I
S THE END-OF-FILEMARKER FOR
THE PROGRAM.": :
```

```
150 PRINT "  IN THIS UTILITY
, THE LEFT-MOST COLUMN INDIC
ATES THE   ASCII CODES, AND
THE NEXT   COLUMN GIVES SOME
 INDICATIONOF THEIR MEANING.
": : :

160 GOSUB 370

170 FOR I=129 TO 254 :: READ
 C$(I):: NEXT I

180 PRINT :: INPUT "NAME OF
(MERGE FORMAT) FILE? ":N$ ::
 CALL CLEAR :: OPEN #1:"DSK1
."&N$,VARIABLE 163 :: DISPLA
Y AT(1,4):"ASCII CODE FOR LI
NE"

190 IF L=65535 THEN 340

200 LINPUT #1:X$ :: L1=ASC(S
EG$(X$,1,1)):: L2=ASC(SEG$(X
$,2,1)):: L=L1*256+L2

210 DISPLAY AT(1,24):STR$(L)
:: C=1 :: J=2

220 S1=LEN(STR$(L1)):: DISPL
AY AT(3,4-S1):STR$(L1)&" "&S
TR$(L1)&"*256" :: S2=LEN(STR
$(L2)):: DISPLAY AT(4,4-S2):
STR$(L2)&"  +"&STR$(L2)

230 FOR I=3 TO LEN(X$):: R=I
+2+2*(C-1):: IF I>40 THEN R=
R-16 :: IF I>80 THEN R=R-16
:: IF I>120 THEN R=R-16

240 J=J+1 :: Y(I)=ASC(SEG$(X
$,I,1)):: DISPLAY AT(R,C+3-L
EN(STR$(Y(I)))):STR$(Y(I))

250 IF Y(I-2)=201 THEN IF Y(
I-1)=0 THEN DISPLAY AT(R,C+4
):STR$(Y(I)):: GOTO 300 ELSE
 DISPLAY AT(R,C+4):"+"&STR$(
Y(I)):: GOTO 300

260 IF Y(I-1)=201 THEN IF Y(
I)=0 THEN 300 ELSE DISPLAY A
T(R,C+4):STR$(256*Y(I)):: GO
TO 300

270 IF Y(I)>254 THEN DISPLAY
 AT(R,C+4):"*"

280 IF Y(I)>128 AND Y(I)<255
 THEN DISPLAY AT(R,C+4):C$(Y
(I))

290 IF Y(I)>31 AND Y(I)<91 T
HEN DISPLAY AT(R,C+4):CHR$(Y
(I))

300 IF J<20 THEN 330

310 C=C+14 :: J=0 :: IF C>16
 THEN C=1

320 IF I=40 OR I=80 OR I=120
 THEN 350

330 NEXT I :: IF I=LEN(X$)+1
 THEN 350

340 CLOSE #1 :: CALL CLEAR :
: END

350 GOSUB 370

360 IF I<>LEN(X$)+1 THEN 330
 ELSE 190

370 DISPLAY AT(24,3)BEEP:"PR
ESS ANY KEY TO GO ON"

380 CALL KEY(0,K,S):: IF S=0
 THEN 380

390 FOR CL=3 TO 24 :: DISPLA
Y AT(CL,1):: NEXT CL :: RETU
RN

400 DATA "::","!",IF,GO,GOTO
,GOSUB,RETURN,DEF,DIM,END,FO
R,LET,BREAK,UNBREAK,TRACE,UN
TRACE,INPUT,"DATA",RESTORE,R
ANDOMIZE

410 DATA NEXT,READ,STOP,DELE
TE,REM,ON,PRINT,CALL,OPTION,
OPEN,CLOSE,SUB,DISPLAY,IMAGE
,ACCEPT,ERROR,WARNING,SUBEXI
T,SUBEND,RUN

420 DATA LINPUT,,,,,,THEN,TO
,STEP,",",;,:,),(,&,,OR,AND,
XOR,NOT,=,<,>,+,-,*,/,^,,,,,
EOF,ABS,ATN,COS,EXP,INT

430 DATA LOG,SGN,SIN,SQR,TAN
,LEN,CHR$,RND,SEG$,POS,VAL,S
TR$,ASC,PI,REC,MAX,MIN,RPT$,
,,,,,,NUMERIC,DIGIT,UALPHA,S
IZE,ALL

440 DATA USING,BEEP,ERASE,AT
,BASE,,VARIABLE,RELATIVE,INT
ERNAL,SEQUENTIAL,OUTPUT,UPDA
TE,APPEND,FIXED,PERMANENT,TA
B,#,VALIDATE
```

## PC NOTES

Our TI PC program for this month is a little File Menu and Loader program written in Basic (See listing below). When you run this Loader a screen with your program files, similar to diagram 1, will be displayed on your monitor. You can then use the cursor arrow keys to move around the menu selections. When you are on the program that you want to run just press RETURN and the program will boot in and run. If you want to look at the other drive move the indicator down to B:*.BAS and press RETURN. This will display the Basic program files from drive B.

You cannot RUN the files with the .DAT, .EXE or .COM extensions from Basic so the program will halt with an error if you try. If you have a hard disk drive you might want to change the B 's in line 5 to E 's. If you have external drive(s) you can easily add another line similar to lines 4 or 5 to accommodate the drive(s). The easiest way to accomplish this is to:

1. RENUM the program.
2. Press F1 RETURN to List the program.
3. Move the cursor up to line 50 and change the line number to 55.
4. Change the B 's to the proper drive letter and then press RETURN.
5. Now press ESC and type in RENUM 1,,1 and press RETURN.
6. LIST the program to check the changes and then SAVE"FILEMENU

You might want to add the following code to the very beginning of line 1.

```
KEY 12,CHR$(27)+"RUN FILEMENU"
```

Then whenever you press F12 and RETURN (as a fail safe), FILEMENU will boot in and run. You can also type in BASIC FILEMENU and press RETURN from MS-DOS and Basic and Filemenu will boot in from the system and run. You could also create a batch file with EDLIN.COM that would do the same thing from MS-DOS.

A few last notes on the program for this month. Be very careful of the spacing on lines 4, 5 and 6 when you type them in or the program won't work right because the indicator is not allowed to move to a space character. If you change the screen display around make sure that the LOCATE in line 3 matches the CHR$(SCREEN(.... in line 14. This is what tells the program which drive you want to load and run from. The [ QUIT ] selection puts you into the Basic command mode and [ SYSTEM ] returns you to MS-DOS. Try changing the COLOR's on lines 2 and 11, and you can delete PALETTE from line 2 if you don't have a graphics board, have fun.

<u>Diag 1.</u>

```
                                A:*.BAS

FILEMENU.BAS CHAR    .BAS CHAR-ASC.BAS BOXES   .BAS BOXES1  .BAS PATTERN2.BAS
BOXES2  .BAS PATTERN1.BAS PAT1ASC .BAS ALLPAT  .BAS BOXASC  .BAS FLMEN   .BAS
A:*.BAS      A:*.*        A:*.DAT      A:*.EXE      A:*.COM
B:*.BAS      B:*.*        B:*.DAT      B:*.EXE      B:*.COM
[  QUIT  ]   [ SYSTEM ]
```

<u>Diag 2.</u>

```
1 '<< BASIC Program Loader >> V831120 Craig Miller MILLERS GRAPHICS
2 PALETTE: ON ERROR GOTO 17: DEFINT A-Z: COLOR 4,0,0,0: F$="A:*.BAS": KEY OFF
3 X=1: Y=4: R=0: C=0: CLS: LOCATE 2,31,0: PRINT F$: PRINT: FILES LEFT$(F$,7)
6 PRINT "A:*.BAS      A:*.*        A:*.DAT      A:*.EXE      A:*.COM"
7 PRINT "B:*.BAS      B:*.*        B:*.DAT      B:*.EXE      B:*.COM"
8 PRINT "[  QUIT  ]   [ SYSTEM ]": GOTO 12
9 K$=INKEY$: IF K$="" THEN 9 ELSE IF K$=CHR$(13) GOTO 14
10 A=ASC(RIGHT$(K$,1)): R=(A=72)-(A=80): C=13*((A=75)-(A=77))
11 IF X+C<1 GOTO 9 ELSE IF SCREEN(Y+R,X+C)=32 GOTO 9 ELSE LOCATE Y,X: PRINT F$;
12 F$="": Y=Y+R: X=X+C: FOR A=0 TO 11: F$=F$+CHR$(SCREEN(Y,X+A)): NEXT
13 COLOR 0,5: LOCATE Y,X: PRINT F$;: COLOR 4,0: GOTO 9
14 IF MID$(F$,2,2)=":*" GOTO 3 ELSE IF LEFT$(F$,8)="[ SYSTEM" THEN SYSTEM
15 IF LEFT$(F$,7)="[  QUIT" THEN CLEAR: KEY ON: END
16 LOCATE ,,1: RUN CHR$(SCREEN(2,31))+":"+F$
17 COLOR 7,0: LOCATE 12,19: PRINT "F I L E   N O T   F O U N D   E R R O R"
18 FOR A=1 TO 1000: NEXT: RUN
```

# 5TH 1- =FORTH

## CONFIGURING TI FORTH TO YOUR SYSTEM

Now that TI Forth has arrived at the User's Groups lets get started by configuring it to match your own system. I strongly recommend that you read through the ENTIRE Forth manual before you start, it could keep you out of trouble later on. TI Forth is originally configured for 1 single sided disk drive and the printer is set up as RS232.BA=9600 and it has a syntax error on the printer output screen, 72.

I am probably already too late but, BEFORE YOU DO ANYTHING WITH TI FORTH MAKE A BACKUP COPY OF THE DISK !!!! The Disk Manager can be used to backup the TI Forth system diskette (only use single sided). There are a few commands that could be detrimental to your Forth diskette if you are not careful such as CLEAR, UPDATE, FLUSH, ED@ and EDIT. Forth has a 5 screen buffer, 5K, which resides at >2010 through >3423. When this buffer is full and one of the screens in the buffer has been marked as updated Forth will automatically write out to the disk, which could write over something you wanted to keep. So ALWAYS keep the write protect sticker on your original diskette and only use it to make copies of for your use.

Now that you have a backup copy lets configure it to your system. First lets load Forth by following the instructions in chapter 1 of the Forth manual. By now you should have read chapter 3 on the EDITOR so lets put it to use. After Forth boots in you will have a menu of additional vocabulary blocks listed on the screen and the cursor will be below the line that reads TI FORTH. To invoke the EDITOR you must first load the vocabulary for it. This is done by typing in -EDITOR and pressing ENTER. Forth will then go out to drive 1 and load and compile the appropriate SCREENs into memory.

After the cursor comes back you can then EDIT the various SCREENs on your BACKUP DISKETTE!!! To edit a SCREEN just type in the SCREEN number followed by a space and the word EDIT and then press enter. The editor will take over from there and display the SCREEN.

So now lets modify the printer section of Forth and fix the syntax error. Type in EMPTY-BUFFERS DECIMAL and press ENTER. This will clear out the buffer and set the number base for decimal numbers. Now type 72 EDIT and press ENTER. The 72nd SCREEN has the RS232 information for your printer. For a parallel printer replace RS232 and RS232.BA=9600 as indicated by the under lined PIO's in the listing below. If your printer is serial but at a different baud rate or uses .LF or .EC etc just insert or delete the appropriate characters as needed on line 4 after " RS232. The syntax error is on line 5. To fix it change PAB_ADDR @ VDP to PAB-ADDR @ VDP. This applies to the 01NOV82 disk name version only, the TI-FORTH disk name version is OK.

```
 0 ( ALTERNATE I/O SUPPORT FOR  PIO  PNTR 12JUL82 LCT)
 1 0 CLOAD INDEX     BASE->R DECIMAL 68 R->BASE CLOAD STAT
 2   0  0  0  FILE >PIO    BASE->R HEX
 3 : SWCH >PIO   PABS @ 10 + DUP PAB-ADDR ! 1- PAB-VBUF !
 4   SET-PAB OUTPT F-D" PIO"                            OPN 3
 5   PAB-ADDR @ VSBW 1 PAB-ADDR @ 5 + VSBW  PAB-ADDR @ ALTOUT ! ;
 6 : UNSWCH 0 ALTOUT ! CLSE ;
 7 : ?ASCII ( BLOCK# --- FLAG )
 8        BLOCK 0 SWAP DUP 400 + SWAP
 9        DO I C@ 20 > + I C@ DUP 20 < SWAP 7F > OR
10           IF DROP 0 LEAVE ENDIF LOOP  ;
11 : TRIAD 0 SWAP SWCH 3 / 3 * DUP 3 + SWAP
12   DO I ?ASCII IF 1+ I LIST CR ENDIF LOOP
13   -DUP IF 3 SWAP - 14 * 0 DO CR LOOP
14   OF MESSAGE OC EMIT  ENDIF UNSWCH  ;
15 R->BASE   -->
```

After you have made the proper changes press FCTN 9 (BACK) and you will be out of the edit mode. Then type in FLUSH and press ENTER and the changes will be written out to the disk. Now type in MENU and the original menu will come on the screen next type in : XXX ; and press ENTER. To load the printer vocabulary type in -PRINT and press ENTER.

To check your printer turn it on and then type in SWCH 72 LIST UNSWCH and press ENTER. When you do this you should get a printout of the revised SCREEN 72. SWCH tells Forth to change from screen output to printer output, 72 LIST lists out SCREEN 72 and UNSWCH changes from printer output to screen output. If it didn't work right then type in FORGET XXX and press ENTER. This erases the printer vocabulary out of memory so you can reload the new one after you reedit SCREEN 72.

After you have the printer working right type in COLD and press ENTER. This will re-BOOT Forth and start it up with just the standard vocabulary. WARNING what we are about to do will write over part of the standard screens, you might want to make a backup copy of your disk with the new printer selection first. Before we go too far here may I suggest that you load -64SUPPORT and edit SCREEN 72 to see the difference between the 64 column editor and the standard one. You will need to decide which editor you like better since you can only have one in memory and we will be setting it up to auto-boot. Now type in COLD and press ENTER. Next you will need to decide which of the menu selections you will be using the most so we can load them into memory. I might suggest that you load the following selections for use during your introduction to TI Forth. Type in the following line and press ENTER.

-GRAPH -DUMP -VDPMODES -COPY

If you are going to use the ASSEMBLER portion of Forth type in -ASSEMBLER and -CRU and press ENTER. Now type in : yourname ; and press ENTER. This will place your name in the dictionary below the print and edit vocabulary so you can easily FORGET these words to open up memory space before you load your programs. If your programs will be using the printer or the floating point routines then load in -PRINT and/or -FLOAT before you enter your name. Next load in -BSAVE and -EDITOR or -64SUPPORT, you have now loaded most of Forth's vocabulary so you can easily become familiar with it. After you learn more about Forth you may want to be more selective on what is auto-booted in.

Now what we are going to do is to set up Forth to automatically boot in all of the menu options that you would like in a binary fast loading fashion. Type in

' TASK 51 BSAVE

(NOTE: ' is an apostrophe - FCTN O )
and press enter. This will save everything between HERE and TASK in the dictionary as a binary image on the disk, starting with screen 51. Odds are that it will write over screens 51 through 64 or more but it won't matter because they are saved in the binary image. If you load the entire vocabulary except -TRACE there will be approximately 3900 bytes free for your use.

Now that you have executed BSAVE you will need to modify SCREEN 3, the auto-boot SCREEN. Since you already have the editor loaded you can just type in EMPTY-BUFFERS and then type in 3 EDIT. When we saved the vocabulary with BSAVE we eliminated the need for much of what is on SCREEN 3. After you have modified your SCREEN 3 to match the one listed below press FCTN 9 (BACK) and then type in FLUSH. You have just completed setting up Forth to rapidly load your menu selections and to match your printer output. To test it out just type in COLD and press ENTER and Forth will rapidly boot the vocabulary that you BSAVEd.

NOTE: To set up for your disk drive types just delete the ( from the beginning of the line(s) that describe your drive(s). If you have a single sided and double sided drive they will have to be configured as single sided. If look at APPENDIX F in your Forth manual you will notice that many of the selections we loaded also loaded other selections. On SCREEN 3 we also added 2 new words to Forth, FREE returns the available memory for your program and PAGE clears the screen and homes the cursor to the upper left hand corner.

```
 0 ( WELCOME SCREEN ) BASE->R  HEX  10 SYSTEM   ( Clears Screen )
 1 0 0 GOTOXY ." Loading.. TI Forth " CR  10 83C2 C! ( QUIT off )
 2 DECIMAL 51 BLOAD  16 SYSTEM   MENU
 3
 4  1 VDPMDE  !               ( Tells Forth you're in TEXT Mode )
 5  0 DISK_LO !               ( Allows EDIT/COPY on all SCREENS )
 6
 7 ( 180 DISK_HI !            ( Set up for 2 Single Sided Drives )
 8
 9 ( 180 DISK_SIZE !          ( Set up for Double Sided Drive{s} )
10 ( 360 DISK_HI !            ( Set up for 2 Double Sided Drives )
11
12 : FREE SP@ HERE - . ;     ( Displays amount of free memory )
13 : PAGE 0 0 GOTOXY CLS ; ( Clears Screen & Homes Cursor )
14
15 R->BASE
```

If you have a hard time reading the white letters on the blue background you can change them to the Extended Basic colors by including the following commands on any of the blank lines on SCREEN 3.

23 7 VWTR

and the screen will change to cyan with black letters. Next month we will start looking at programming in Forth. Until then have fun and read the manual and "Starting Forth" by Leo Brodie.

## S U B S C R I P T I O N   I N F O R M A T I O N

**THE SMART PROGRAMMER** - a monthly 16+ page newsletter published by **MILLERS GRAPHICS**
U.S. 12.50 year - Foreign Surface Mail 16.00 year - Foreign Air Mail 26.00 year

To subscribe send a Check, Money Order or Cashiers Check, payable in U.S. currency

TO: **MILLERS GRAPHICS**
**1475 W. Cypress Ave.**
**San Dimas, CA 91773**

MG

**MILLERS GRAPHICS**
1475 W. Cypress Ave.
San Dimas, CA 91773

BULK RATE
U.S. POSTAGE
PAID
San Dimas, CA 91773
PERMIT NO. 191

JAMES A. COURTNEY 85-01BR
RR #4 2744 W. HUME RD.
CRIDERSVILLE, OH
45806

*THE SMART PROGRAMMER*