MILLERS GRAPHICS

MAY 1984

$1.50

THE SMART PROGRAMMER

Once again I would like to apologize for the tardiness of the newsletter release. I also want to thank EVERYONE for being so patient and understanding. Thanks to the diligent efforts of Steve Mildon, we have recently completed the main project we were working on with CorComp Inc.. So, now we can devote our main effort to getting the newsletter back on track and back out on a regular basis. A few people have asked if we are still going to release full subscriptions of the newsletter and the answer is definitely YES. We will not skip a single issue and we will not combine two months into one 16 page newsletter.

This month we have an extensive memory map of Extended Basic's use of VDP RAM. We have also mapped out the Disk File Buffer area as it sits in memory for CALL FILES(3). So lets get started.

In doing some recent research we discovered that when you OPEN the RS232 card for input or output that most of the software and hardware switch options can be spelled out. Such as:

```
OPEN #1:"RS232.BAUD RATE=9600.CHECK PARITY"
```

When the RS232 DSR parses the OPEN statement it only checks the first two characters, then it looks for a space or = sign and then it takes in the appropriate characters up to the next space period or close quote. Also, the second and up .(switches) can have spaces between the last switch and the next . such as:

```
OPEN #1:"PIO.EC  .TW  .LF"
```

It is also possible to OPEN the same port more than once at the same time. However, the Hardware switch options must be the same but, the Software switch options may be different.

The Hardware switch options are;

Baud Rate, Data Bits, Parity and Two Stop Bits.

The Software switch options are;

Nulls, Check Parity, Echo, Carriage Returns and Line Feeds (.CR) and Line Feeds (.LF)

The last item discovered was that the .CR and .LF options have no effect when FIXED length records are specified for a PRINT command. The exact number of characters set up in the length is output exactly as they are sent to the RS232 Card.

Over the past years there have been many times that we wanted to test a program or routine without using Memory Expansion. Each time we wanted to do this we had to shut down our system and remove the card from the P E Box. This proved to be a real pain in the neck, so we finally decided to sit down and write a program that would shut off Memory Expansion for us.

After a short time of searching for the proper combination of CALL LOAD's we came to the conclusion that we were fighting the values stored in VDP RAM (>0370 - >03BE). This area holds pertinent system data and as such would also require new values to be POKEVed into it. That, we decided was not a good solution because it required that we load the PEEKV and POKEV routines each time we wanted to shut off the memory.

With a little more playing around the following solution came to be. First type in the three following programs and save them to disk using the names indicated. Then when you want to turn off the memory type in RUN "DSK1.MEMOFF". To turn it back

on type in RUN "DSK1.MEMON". The key to the solution was to immediately load another program after the CALL LOAD is executed. You MUST either OLD or RUN the program you cannot MERGE it!!!

When you OLD or RUN a program Extended Basic looks at the address we changed and sets up the proper pointers throughout the rest of CPU and VDP memory. MERGE, however, does not change these pointers and the system gets totally confused. If you try to LIST or RUN a MERGEd program your system will probably lock up and you will have to shut it off and then back on.

The other key is to keep at least one line of a program in VDP memory when Memory Expansion is shut off. The 1 REM was added to the MEMTEST program to allow you to delete line 100, without turning on the Expansion Memory, so you could type in your own program. To turn the memory back on you can either run the MEMON program our you can type in NEW or just delete all of the program lines form memory.

The Expansion Memory will also come back on if you create an IO ERROR when you access the disk, such as typing in a filename that does not exist on the disk. Typing in the command SIZE will verify that the memory is either ON or OFF. Even though you have turned off Expansion Memory CALL LINK & CALL LOAD will work fine. This is because the validation code is still sitting in Low Mem-Exp (after executing CALL INIT). What this means is that you can fill up Mem-Exp with Assembly routines, turn off Mem-Exp and then load an Extended Basic program into VDP RAM that calls these routines. Here are the three programs for turning the memory ON and OFF (don't forget to execute CALL INIT):

```
MEMOFF
     100 CALL LOAD(-31868,0,0)::
     RUN "DSK1.MEMTEST"

MEMON
     100 CALL LOAD(-31868,255,231
     ):: RUN "DSK1.MEMTEST"

MEMTEST
     1 REM
     100 CALL CLEAR :: CALL PEEK(
     -31952,A):: IF A=55 THEN PRI
     NT "Exp-Memory is OFF" ELSE
     PRINT "Exp-Memory is ON"
```

OOPS!

In the opps column this month we want to try and answer few problems that some people are having with past newsletters.

In an earlier issue on FORTH we stated that you can copy the FORTH disk with the Disk Manager Module. It is VERY important that when you use the Disk Manager to copy FORTH that you format the disk as Single Sided. If you do not the menu and a few other screens end up on side two and as such they will not work properly. TI-FORTH accesses the disk using sector IO not file IO like the other languages. Due to the nature of the Disk Controller DSR, Forth's boot screen (screen 3) is not sitting at sectors 12-15 when you copy the FORTH disk with the Disk Manager onto a Double Sided disk it ends up on sectors 367-370 (side two) and Forth's Screen 3 is left blank.

A few people have written in stating that they are having problems with the BSAVE for Forth from the March issue. Part of this may be because of the above paragraph. Part of it may be due to one or more of the following items:

1. Be sure you are in DECIMAL base.
2. Be sure you are typing in ' TASK 51 BSAVE and then enter. The Apostrophe (FCTN 0) is VERY important!!
3. Be sure your Screen 3 has been modified to match the one shown on page 15 in the March issue. If you type in the screen EXACTLY as shown you will be set up for one single sided disk drive. Make sure you have 51 BLOAD and not 51 LOAD on line 2.
4. In order to access BSAVE you must load it from the Forth menu by typing in -BSAVE and pressing enter.

Its quite hard to analyze the exact nature of the problem without actually seeing what is going wrong. I hope these two paragraphs are some help.

In the April issue we dropped a very important character from the Assembly listing of the PEEKV, POKEV & POKER routines. Line 28 on page 7 should read;

0028 0014 MYWS BSS >10

That should clear up any problems you <u>might</u> have had when using the Assembler version. The CALL LOAD version works fine.

A few people have written stating that they are having problems loading the programs after they have been Assembled. The problem they stated was that they keep getting UNRECOGNIZED CHARACTER error messages when they try and load it through Extended Basic. The Extended Basic loader will not load compressed object code, so when the assembler asks for OPTIONS do not type in C. This will assemble the program in uncompressed format and it will load and LINK to XB fine. See page 33 (Options) and pages 410-419 (Comparisons with TI Extended Basic Loader) in the Editor Assembler manual for additional information.

In the February issue we stated that you could use the number conversion program, that was printed in the Free Mini Sample of the Smart Programmer, to convert the Assembly Opcode Values into CALL LOAD values. Well it seems that few people are having some problems with this so lets take another look at it. The Opcode Value is made up of two bytes, one word, and the CALL LOAD values must be one byte. The number conversion program converts word hex values into word decimal values and it also split them up into two byte values. The Opcode Value for the instruction MOV R2,R1 is >C042. This value converted to decimal in two's compliment is -16318 and converted to an unsigned number is 49218. To get the CALL LOAD values just split the hex value into two bytes >C0 and >42. Now >C0 equals decimal 192 and >42 equals decimal 66. So the CALL LOAD to load this instruction would be CALL LOAD(address,192,66). If you multiply 192 * 256 and add 66 to it you get 49218. I hope this has helped a little.

We have received many letters and questions and we appreciated all the comments, tips and compliments. Some letters have contained self addressed stamped envelopes and we are sorry that time has not permitted us to reply directly. We have read and filed all of these letters into different question categories and we plan to answer as many as possible in the pages of the Smart Programmer. So please don't send self addressed envelopes with your questions. Also if you are having a problem with something please include the ATA or LTA number from the bottom of your console and Extended Basic module, this may help us narrow down some of the different operating system differences.

Q & A

Over the past few months we have received some questions regarding the Power Up routine and its sequence of events.

The Power Up routine resides in console ROM and console GROM chip 0, with the majority of it in GROM. Listed below is the sequence of events, as we know it, that take place from the moment power is applied to the console.

1. When power is first applied to the 9900 microprocessor it executes a level 0 interrupt which is a reset. When this happens it knows to grab its workspace pointer from address >0000 and its program counter or instruction pointer from address >0002. In the 99/4A these addresses are burned into the console ROM. Once it has these addresses it begins to execute the code that is pointed to by address >0002.

2. The code pointed to by address >0002 loads R13 with >9800, the GROM read address, R14 with >0100, the status flag and R15 with >8C02, the VDP write address.

3. The balance of the Power Up routine is in GROM chip 0. First it clears out the sound list indicator at >83CE and then it turns off the Speech Synthesizer, if you have one, and turns off the sound generators.

4. Next it initializes the Data and Subroutine GPL stacks in Scratch Pad RAM.

5. Then it loads the VDP registers with default values stored in a GROM table.

6. Most of the Scratch Pad RAM is then cleared out by writing zeros to it.

7. The keyboard interrupts are then disabled and the cassette audio gate is turned on, this allows you to hear the cassette through your monitor speaker.

8. Next the VDP and external interrupts are enabled.

9. The cassette motors are then enabled, this sets up the remote jack so the motors can run.

10. Next it generates a BEEP sound by CALLing the TON1 routine in GROM chip 0.

11. And then it determines the size of VDP RAM, 4K or 16K, and sets bit 0 in VDP register 1 to 1 for 16K. I guess that TI thought they might produce a 4K console at one point in time.

12. The first 4K of VDP RAM is then cleared out and the default color and character tables are moved from GROM to VDP RAM.

13. Next the keyboards are initialized by scanning them. ie: CALL KEY(5.. ,CALL KEY(4.. , CALL KEY(3.. etc.

14. Then it moves the data for the Title Screen into the screen image table. At this point the screen can not be seen because it is turned off. Bit 1 in VDP register 1 is set to zero.

15. The ROMs(DSR) and GROMs are then searched for power up headers and if they have one they are executed.

16. After the search and execution of power up headers is complete the Title Screen is displayed by writing a 1 to bit 1 of VDP register 1. This is the first time that you are able to see the Tile Screen.

17. Next it initializes the Random Number Generator and then waits for you to 'PRESS ANY KEY'. After you press a key it generates another BEEP sound.

18. And then it searches through the cartridge ROM and GROM for application programs and builds a list of their names. ie: Editor/Assembler, Extended Basic, Basic etc. Note: some consoles, V2.0 (C) 1983, do not search ROM only GROM.

19. At this point the screen is once again turned off, VDP register 1, bit 1 is set to zero and then it sets up the menu screen with the application programs it found.

20. After the menu is set up it turns on the screen by writing a 1 to bit 1 of VDP register 1, and waits for you to make a selection.

21. If you make an illegal selection it generates a HONK sound, otherwise it sets up the starting address for the application program you selected and branches to it.

We have received a number of questions regarding the TMS 9918A Video Display Processor chip and where to get more info and technical data on it.

TI publishes Data Manuals on every chip they produce. These manuals range in price up to about 15.00. They can usually be obtained through a good electronics supplier, a local TI sales office or exchange center or direct from the TI Semiconductor Group. Most of the TI sales offices are listed in the phone book and the address for the TI Semiconductor Group is:

Texas Instruments
P.O. Box 1443
Houston, TX 77001

You can probably write to them and find out how much the TMS9918A/TMS9928A/TMS9929A Video Display Processors Data Manual #MP010A costs.

We have received a number of question on the care and maintenance of diskettes and disk drives.

First off in most of the information that I have read on floppy diskettes I've noticed that they forget to mention one very important thing. I, like many other people, have the BAD habit of laying floppies on top of my desk without putting them back into their protective sleeves. I'm quite lucky that I haven't damaged one yet. You see, side 1 on the diskette is the side opposite the label side. This is the side that comes in contact with the desk and anything on it when you lay down the floppy without its protective sleeve. So, DON'T get into my bad habit, put your disks back into their sleeves as soon as they come out of the drive.

I have also read a lot of articles about making floppies into flippies (punching new holes in the jacket to use the other side of the disk in a single sided drive). My opinion on this subject is **DON'T.** On a single sided drive the head is on the bottom and there is a felt type pressure pad on the top that holds the floppy against the head. After a few months of use this felt pad collects dirt and dust. When you flip your diskettes over you are allowing this dirty felt pad to come into a pressure type contact with your valuable

data. It only takes one scratch on sectors 0 or 1 to wipe out an entire diskette!!! Also when you flip your disks over you are causing them to rotate backwards in the drive. Inside the jacket is some special fabric that helps keep the diskette clean. But, when rotate it backwards you may cause the pads to release some of the dirt they collected right onto your diskette. Think of it like a clothes brush you've just used to brush your sweater with and then try and brush your black pants off, only reverse the brush stroke direction - what a mess! I know some people will send in letters stating that they have been using flippies for a while now without any trouble. My only comment is that I would rather spend the extra couple of bucks for another floppy than retype in 90K of Data and Files.

You should store your floppies in a covered upright position not laying down. The original box works fine or one of the many floppy files is also a good idea. When they are stacked laying down the fabric in the jacket is pressed against the diskette and it may imbed some dirt into the surface and wipe out some data.

My opinion on cleaning your disk drives with a head cleaning diskette is only use it as a last resort! If you have made sure that its not the floppy, a bad connection or an improperly closed file then run the destructive disk test. If you have a lot of errors on a lot of different floppies then as a last resort use the head cleaner BUT follow the directions to the letter!! If they say 9 or 10 drops don't use 20 or 30 unless you want it all over the inside of your drive. If they say to run the disk for 30 seconds then don't run it any longer!

The reason I'm against these head cleaners it that they are ABRASIVE and if you use them too much you will wear out the read/write head in your drive. I know the head cleaning kit manufactures say to clean your drive at least once a week & more often if they are used heavily, but nowhere on my kit does it guarantee NOT to hurt my drives. As a matter of fact most of them have a disclaimer and they will only replace the cleaning kit if something goes wrong. We use our computers between 6 and 15 hours a day, 6 to 7 days a week and we don't clean the heads any more than ONCE or TWICE a year.

GPLLNK ROUTINE

Over the past few months we have received a number of requests for a GPLLNK Routine that works with Extended Basic. Also during this same time period we have received a few modified GPLLNK routines from various readers, thank you. The GPLLNK routine we have included in this issue is very similar to the one that TI sends out, except this one works, and I believe that it was sent to us from John Brown.

Once again we have included the Assembly version and the CALL LOAD version for Extended Basic. In the Assembly version there are two programs, GPLLNK & EXEGPL. The GPL program is all you need to add to other Assembly programs in order to use the GPL Routines in GROM chip 0. The EXEGPL program allows you to CALL LINK("EXEGPL",#) to execute a GPL Routine from Extended Basic. The CALL LOAD version contains both the GPLLNK and EXEGPL so you can LINK to the routines. The list of available routines is as follows:
(Note: I believe that >10, >12, >1A, >1C & >1E are intended for use in a GPL program only.)

Hex Dec Name

>10 (16) LINK
Links to subprograms and DSR's.
>8356 points to name location in VDP RAM VDP location contains one byte for length of name then the name. The DSR or subprogram should return back to GROM through RETN. In GPL code a Data byte of >8 immediately following the CALL indicates a subprogram link and >A indicates a DSR link.

>12 (18) RETN
Return from a subprogram or DSR.

>14 (20) CNS or STR
Convert a number into a string.
>834A contains the number in radix 100 notation. >8355 contains 0 for TI Basic format. If >8355 is greater than zero then the number is returned in FIX mode and >8356 must contain the number of significant digits, >8357 must contain the the number of digits to the right of the decimal point. The location of the string is pointed to by adding >8300 to the byte at >8355. >8356 contains the length of the string.

>16 (22) CHAR1
Load 8 dot high Character set.
Loads the Title screen chars into the VDP address pointed to by >834A.

>18 (24) CHAR2
Load 7 dot high character set.
Loads the standard chars into the VDP address pointed to by >834A.

>1A (26) BWARN
Warning message from Basic subprogram in GPL.

>1C (28) BERR
Error message from Basic subprogram in GPL.

>1E (30) BEXEC
Begins execution of GROM Basic program.

>20 (32) PWRUP
Restarts the system by executing the GROM portion of the power-up routine.

>22 (34) INT
Converts a floating point number into its greatest integer value. >834A contains the floating point number in radix 100 and the greatest integer value is returned to >834A.

>24 (36) PWR
Raise a number to a power. >834A contains the power value in radix 100 and >835C contains the value of the number to be raised in radix 100.

>26 (38) SQR
Calculates the square root of a number. >834A contains the input value and the square root is returned to >834A, both in radix 100 notation.

>28 (40) EXP
Calculates the inverse natural logarithm of a number. >834A contains the input value and the result is returned to >834A in radix 100.

>2A (42) LOG
Calculates the natural logarithm of a number. >834A contains the input value and the result is returned to >834A in radix 100.

>2C (44) COS
Calculates the cosine of a number. >834A contains the input and result.

>2E (46) SIN
Calculates the sine of a number. >834A contains the input value and the return value in radix 100.

>30 (48) TAN
Calculates the tangent of a number. >834A contains the input value and the return value in radix 100.

>32 (50) ATN
Calculates the arctangent of a number. >834A contains the input value and the return value in radix 100.

>34 (52) TON1
Generates a BEEP sound.

>36 (54) TON2
Generates a HONK sound.

>38 (56) Get String Space.
Allocates memory space in VDP RAM with a specified number of bytes for a string or PAB. The word at >830C contains the number of bytes to allocate. The word at >831A points to the first free address in VDP RAM and the word at >831C points to the start of the space to be allocated.

>3B (59) BITRVR
This is a bit reversal routine. It will reverse the bits in each byte of a block of specified VDP RAM, ie: make the characters backwards so b becomes d etc. The word at >834A points to the starting address in VDP RAM and >834C contains the number of bytes to be reversed. Unfortunately this routine destroys addresses >8300 through >8340 so it can not be called from Basic or Extended Basic. You can however write and Assembly language routine to save this area of memory, use the routine and then restore this area and return to Extended Basic.

>3D (61) Cassette DSR.
Accesses the cassette DSR. You must set up a PAB before calling this routine.

>4A (74) CHR3
Load the 7 dot high small cap characters. These are the ones used for lower case letters. This routine is in the 99/4A only. >834A points to the starting address in VDP RAM where the characters are to be loaded.

```
*--------------------------------------*
* XB GPLLNK Routine - Save as source   *
* code and add into desired Assembly   *
* program with the COPY directive.     *
*--------------------------------------*
UTILWS EQU  >2038
SUBSTK EQU  >8373
GRMRA  EQU  >9802
GRMWA  EQU  >9C02
GPLWS  EQU  >83E0
PAD    EQU  >8300
*--------------------------------------*
* GPLLNK workspace,program counter     *
*--------------------------------------*
GPLLNK DATA UTILWS,GPL0
GPL0   MOVB @GRMRA,R1          Fetch GROM address
       SWPB R1
       MOVB @GRMRA,R1
       SWPB R1
       AI   R1,-3              Back up to the XML instruction
       MOVB @SUBSTK,R2         Get the stack pointer
       SRL  R2,8
       AI   R2,PAD
       INCT R2                 Push XML address for return
       MOVB R1,*R2
       SWPB R1
       MOVB R1,@1(R2)
       SWPB R2
       MOVB R2,@SUBSTK
       MOVB *R14+,@GRMWA       Set up address to call
       MOVB *R14+,@GRMWA       Second byte (also adjust return)
       MOV  @>2000,R4          Save current XML link
       LI   R3,GPL1            New XML link
       MOV  R3,@>2000
       LWPI GPLWS              Use GPL/XB workspace
       RT                      Go to routine
GPL1   LWPI UTILWS             Should return here
       MOV  R4,@>2000          Restore original XML location
       RTWP                    and go back to caller
* Use the END statement only if not called by COPY directive.
*      END
```

```
*--------------------------------------*
*    Set up CALL LINK("EXEGPL",#)      *
*--------------------------------------*
       COPY "DSK2.GPLLNK"      NOTE remove END statement from GPLLNK
       DEF  EXEGPL             Set link name in REF/DEF Table
FAC    EQU  >834A              Point to FAC
ADDRSS DATA >400               Sits at >24EC dec 9452 if loaded first
*                              use CALL LOAD(9452,?,?) to cange.
EXEGPL CLR  R0
       LI   R1,1
       BLWP @>200C             Use NUMREF routine
       MOVB @FAC+1,@DAT+1      Set up GPL call number
       MOV  @ADDRSS,@FAC
       BLWP @GPLLNK            Do GPLLNK routine
DAT    DATA >16
       RT
       END
```

GPLLNK CALL LOADS

```
1 !          GPLLNK &
     CALL LINK("EXEGPL",#)

2 ! Note:9452 is the address
   of the word that is moved
   into FAC (>834A)

10 CALL INIT

20 CALL LOAD(16368,69,88,69,
71,80,76,37,186)

30 CALL LOAD(16376,69,88,69,
71,80,76,37,72)

40 CALL LOAD(8194,37,216,63,
240)

50 CALL LOAD(9460,32,56,36,2
48,208,96,152,2,6,193,208,96
,152,2,6,193,2,33,255,253,20
8,160)

60 CALL LOAD(9482,131,115,9,
130,2,34,131,0,5,194,212,129
,6,193,216,129,0,1,6,194,216
,2)

70 CALL LOAD(9504,131,115,21
6,62,156,2,216,62,156,2,193,
32,32,0,2,3,37,60,200,3,32,0
)

80 CALL LOAD(9526,2,224,131,
224,4,91,2,224,32,56,200,4,3
2,0,3,128,4,0,4,192,2,1)

90 CALL LOAD(9548,0,1,4,32,3
2,12,216,32,131,75,37,99,200
,32,37,70,131,74,4,32,36,244
)

100 CALL LOAD(9570,0,22,4,91
,32,56,37,106,208,96,152,2,6
,193,208,96,152,2,6,193,2,33
)

110 CALL LOAD(9592,255,253,2
08,160,131,115,9,130,2,34,13
1,0,5,194,212,129,6,193,216,
129,0,1)

120 CALL LOAD(9614,6,194,216
,2,131,115,216,62,156,2,216,
62,156,2,193,32,32,0,2,3,37,
174)

130 CALL LOAD(9636,200,3,32,
0,2,224,131,224,4,91,2,224,3
2,56,200,4,32,0,3,128,4,0)

140 CALL LOAD(9658,4,192,2,1
,0,1,4,32,32,12,216,32,131,7
5,37,213,200,32,37,184,131,7
4)

150 CALL LOAD(9680,4,32,37,1
02,0,22,4,91,66,252)
```

GPLLNK TEST PROGRAM

```
1 !   GPLLNK test program
   must load GPLLNK assembly
        program first.

10 A$="!""#$%&'()*+,-./01234
56789:;<=>?@ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^_`abcdefghijkl
mnopqrstuvwxyz{|}~"

20 CALL CLEAR :: FOR I=1 TO
10 :: PRINT "PRESS ANY KEY t
o change": : :: NEXT I

30 CALL KEY(0,K,S):: IF S TH
EN 40 ELSE CALL LINK("EXEGPL
",22):: CALL LINK("EXEGPL",2
4):: GOTO 30

40 CALL LINK("EXEGPL",22)::
PRINT "LARGE TITLE SCREEN CH
ARS": : :: GOSUB 80

50 CALL LINK("EXEGPL",24)::
PRINT "REGULAR SCREEN CHARS"
: : :: GOSUB 80

60 CALL LINK("EXEGPL",74)::
PRINT "LOWER CASE CHARS 0123
456789": : :: GOSUB 80

70 GOTO 40

80 CALL KEY(0,K,S):: IF S=1
THEN PRINT A$: : :: RETURN E
LSE 80
```

16K VDP RAM Extended Basic Use

>0000	**VDP SCREEN IMAGE TABLE**	**768 Bytes**
	1 Byte per screen position. Character value offset by >60 (96)	
	(Row-1)*32+Col=Address	
	>02E2=New line Address	
>02FF		
>0300	**SPRITE ATTRIBUTE TABLE**	**112 Bytes**
	4 Bytes per sprite. (room for 28 sprites)	
>036F	| vert pos-1 | horz pos | char #+96 | early clock bit : color |	
>0370	**EXTENDED BASIC SYSTEM AREA**	**128 Bytes**
>0371	Auto Boot needed Flag	
>0372	Line to start execution at	
>0376	Saved symbol table 'GLOBAL' pointer	
>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 temp storage place (FAC12)	
>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 . Used in SUBS	
>039E	Merged temp for PAB 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	
>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 (FAC10)	
>03BC	Old top of memory for Relocating Program / Temp for INPUT	
>03BE	New top of memory for Relocating Program	
>03C0	Temp Roll Out Area 32 Bytes (part of scratch pad RAM is moved here for various operations)	
>03DC	Floating point sign	
>03EF		

>03F0 **PATTERN DESCRIPTOR TABLE** **912 Bytes**
SPRITE DESCRIPTOR TABLE

8 Bytes per character / 114 characters (30-143)

>03F0 = Char 30 (The Sprite Motion Table uses the memory
>03F8 = Char 31 space for character sets 15 & 16)
>0400 = Char 32

>077F

>0780 **SPRITE MOTION TABLE** **128 Bytes**
4 Bytes/Sprite

>07FF | vert velocity | horiz velocity | sys use | sys use |

>0800 **COLOR TABLE** **32 Bytes**
1 Byte/Character set

>081F | foreground color : background color |

>0820 **CRUNCH BUFFER** **160 Bytes**

This area is used while crunching ASCII into token codes.

>08BE

>08C0 **EDIT/RECALL BUFFER** **152 Bytes**

This area holds the info you type in on the command line.

>0957

>0958 **VALUE STACK** (Default Base) **16 Bytes**

>0967 Used by the ROM routines SADD, SSUB, SMUL, SDIV and SCOMP.

>0968 The items in this area move according **11888 Bytes**
to the size of the crunched program.
The system also reserves 48 bytes of this area.

The SYMBOL TABLES are generated (except the PAB) during the Pre-Scan period after you type RUN. The strings are placed in memory when they are assigned (ie: A$="Hello")

Without Mem-Exp	With Mem-Exp
STRINGS	STRINGS
DYNAMIC SYMBOL TABLE & PABS	DYNAMIC SYMBOL TABLE & PABS
STATIC SYMBOL TABLE	STATIC SYMBOL TABLE
LINE NUMBER TABLE	Numeric Values, Line Number Table and Program Space moved to High Mem-Expansion
PROGRAM SPACE (crunched program)	

The Line Number Table and the Crunched Program are saved to disk like they reside in memory for PROGRAM "Memory Image" type files.

>37D7

>37D8	**DISK BUFFERING AREA for 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 (11 for CRU 1100)	
>37DC	Maximum number of OPENed files (>03 default)	
	File Control Block for 1st file OPENed 6 Bytes	**518 Bytes**
>37DD	Current Logical record offset	
>37DF	Sector number location of File Descriptor Record	
>37E1	Logical Record Offset (only used with VARIABLE records)	
>37E2	Drive number (high order bit set = Updated Data Buffer area)	
	File Descriptor Record (brought in from the disk 256 Bytes)	
>37E3	File Name	
>37ED	Reserved (>0000)	
>37EF	File Status Flags (file type and 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 (ie: FIXED 80 or VARIABLE 254 etc.)	
>37F5	# of FIXED length records or # of sectors for VARIABLE length (the bytes are reversed ie: LSB|MSB should be MSB|LSB)	
>37F7	Reserved (>0000 >0000 >0000 >0000)	
>37FF	Pointer Blocks - 6 nibble, 3 byte, clusters that point to the Start Sector numbers and the highest logical Record Offset in the cluster. Change the nibble order from |ss2:ss1| |ro1:ss3| |ro3:ro2| to |ss3:ss2:ss1| |ro3:ro2:ro1|	
>38E3	Data Buffer area 256 Bytes	
>39E3	File Control Block for 2nd file OPENed 6 Bytes	**518 Bytes**
	same pattern as above	
>39E9	File Descriptor Record 256 Bytes	
	same pattern as above	
>3AE9	Data Buffer area 256 Bytes	
>3BE9	File Control Block for 3rd file OPENed 6 Bytes	**518 Bytes**
	same pattern as above	
>3BEF	File Descriptor Record 256 Bytes	
	same pattern as above	
>3CEF	Data Buffer area 256 Bytes	
>3DEF	**VDP STACK AREA**	**252 Bytes**
>3EEA		
	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	
>3EEF	(?? not used any more was for the 99/4 ??)	**6 Bytes**
>3EF4		
>3EF5	**VOLUME INFORMATION BLOCK**	**256 Bytes**
	(Copy of Sector 0 from the last disk accessed for a WRITE)	
>3FF4	Contains Disk Name, type and bit map for used sectors.	
>3FF5	**FILE NAME COMPARE BUFFER**	**11 Bytes**
>3FFF	Contains disk number and 10 character file name for last access.	

COLOR EDITOR

The following program was sent to us from one of our European readers. It allows you to mix any two colors and displays a large block of the new color. After using it for a little while we made a couple of small modifications that allowed it to run a little faster. We left the original code in place with REMs in front of it so you could see the difference. There are many ways to get the computer to do something and its always fun to see how another way works.

This program places 15 double sized sprites down the right hand side of the screen to form the color selections. You are then allowed to move another sprite, the marker, up and down this column to make a selection. When you are on the color you want just press the fire button on the number one joystick or any key but the up and down arrow keys. The program then uses this color as the foreground color and the next color chosen will be the background color. It is fun to play with and it shows you which colors mix well to form a new shade for use in another program. We would like to thank Mr. Reitinger for sending us this program. We also hope he doesn't mind the minor modifications to his program.

```
1 !     COLOR EDITOR
    for mixing any desired
    2 colors with joyst or
         keyborad.

2 !    With Greetings

       E.H.REITINGER
      Vienna, Austria

3 ! TI99-Journal-Klub
        A-1150 Wein
     Felberstrabe 24/26

10 CALL SCREEN(16):: CALL CL
EAR

20 M$="55AA55AA55AA55AA" ::
A=122

30 CALL MAGNIFY(2):: CALL CH
AR(64,RPT$("F",16),34,"FF818
1FFFFFF",128,"FFFFFFFFFFFFFF
",73,M$):: CALL COLOR(3,16,2
,4,16,2,6,1,1,5,2,1)

40 CALL VCHAR(1,27,64,192)::
 CALL HCHAR(23,1,64,162):: H
=1

50 G=-2 :: FOR I=3 TO 16 ::
CALL SPRITE(#I,64,I,(G+I)*12
,230):: NEXT I :: CALL SPRIT
E(#2,34,16,5,230)!#16,128,16
,17,230)

60 CALL SPRITE(#1,42,2,A,231
)

70 FOR S=4 TO 22 :: CALL HCH
AR(S,3,73,24):: NEXT S

80 CALL JOYST(1,X,Y):: ON (S
GN(Y)+2)GOTO 90,130,110

90 A=A+12 :: IF A>170 THEN A
=2

100 CALL LOCATE(#1,A,231)::
GOTO 130

110 A=A-12 :: IF A<0 THEN A=
170

120 CALL LOCATE(#1,A,231)

130 CALL KEY(1,K,S):: IF S=0
 THEN 80

140 IF K=5 THEN 110 :: IF K+
1=1 THEN 90

145 F=INT(A/12+2):: CALL SOU
ND(200,660,2):: GOTO 180 ! T
his was inserted to replace
line 150,160 & 170 Ed.

150 !FOR F=2 TO 16 :: CALL C
OINC(#1,#F,3,C):: IF C THEN
CALL SOUND(200,660,2):: GOTO
 180

160 !NEXT F

170 !GOTO 80

180 CALL COLOR(6,F,H):: DISP
LAY AT(24,9)SIZE(7):USING "
## ## ":F,H :: H=F :: GOTO 8
0
```

5TH 1- =FORTH

We have been receiving a lot of requests lately for some Forth programs that use graphics. Well this month, thanks to Mr. Volk, we have a program called Diamond Draw. This program uses the Bit Mapped mode and allows you to move a diamond around the screen to draw lines in different colors. After you have typed it in and saved it out to disk (FLUSH) you will need to load -VDPMODES and -GRAPH if your Forth system is not set up to boot these in initially. The just type in the first screen number you saved this program on and LOAD. After the program is loaded and complied type in RUN to start the program. The instructions for using Diamond Draw will appear on the screen so just proceed from there. Its a lot of fun to play with. All we need now is a BSAVE routine to save the drawings.

Mr. Volk also sent us a copy of some new words he has added to his Forth system and a copy of a single disk copying routine. Since the disk copy routine uses his new words AT, WORK and TOP (our PAGE) you will need to load in screen # 91 first or add these three words to the beginning of screen # 92. IF your Forth system has been set up using the BSAVE routine from the March issue then -SYNONYMS and -COPY should already be loaded. Once you have LOADed the screen(s) just type in RUN. The program automatically prompts you from there. I might recommend that you cover the write protect notch on the MASTER DISK, just in case you goof and place the master disk in for the copy disk. It probably wouldn't hurt it since it is doing a sector copy but your COPY disk would end up incomplete. With the notch covered the program will halt with an error if you goof and you can just type in run to restart it.

---- Thank you Mr. Volk ----

```
SCR #91
  0 ( MY MOST USED WORDS  by J. Volk)
  1 ( LOAD -SYNONYMS FIRST if not already BLOADed)
  2 : MYLOAD -GRAPH -VDPMODES ; ( Will load these options )
  3 : AT GOTOXY ; ( Same as 'Display At' )
  4 : TOP CLS 0 0 AT ; ( Same as Brodie 's 'PAGE' )
  5 : RANDOM RND 1+ . ; ( n RANDOM >>> gives random number )
  6 : PICK ( Leave copy of n1-th number on top of stack )
  7          ( n1 --- n2 )
  8      2 * SP@ + @ ;
  9 : ROLL ( Rotate nth number to top of stack ) ( n --- n)
 10      DUP 1 = IF DROP ELSE DUP 1 DO SWAP R> R> ROT >R >R >R LOOP
 11      1 DO R> R> R> ROT ROT >R >R SWAP LOOP THEN ;
 12 : TEST BEGIN ." HELLO THERE" 2 SPACES ?TERMINAL UNTIL ; ( FCTN 4
 13  TO END )
 14 : SGN DUP IF DUP 0< IF -1 ELSE 1 ENDIF ELSE 0 ENDIF ;
 15 : WORK BLOCK DROP UPDATE ; ( My word to update a FORTH screen )
```

```
SCR #92
  0 ( A Word to copy FORTH disks-Single  Drive 5/16/84 J. Volk )
  1 ( Load Screen #91 and -COPY then RUN )
  2 0 VARIABLE COPYSCR  0 DISK_LO !
  3 : MES1 COPYSCR @ 88 > IF CLS ABORT ENDIF TOP 2 11 AT ." INSERT M
  4 ASTER DISK           " KEY DROP ; ( PRINT MESSAGE AND KEY PRESS )
  5 : COPY1 5 0 DO COPYSCR @ WORK 2 20 AT ." SCR # " COPYSCR ? 1 COP
  6 YSCR +! LOOP ; ( DO THE WORK AND LET US KNOW-GET NEXT SCREEN )
  7 : COPY2 2 11 AT ." INSERT COPY DISK-ANY KEY      " KEY DROP ;
  8 ( COPY 5 SCREENS AND PRINT MESSAGE )
  9 : GETIT BEGIN MES1 COPY1 COPY2 FLUSH COPYSCR @ 89 = UNTIL ;
 10 ( RUNS ABOVE WORDS )
 11 : MES0 TOP 2 11 AT ." INITIALIZE FORTH DISK ? (Y/N) " ;
 12 : MSG TOP 2 11 AT ." INSERT COPY DISK " KEY DROP ;
 13 : RUN MES0 KEY 89 = IF MSG 0 FORMAT-DISK DISK-HEAD ENDIF GETIT ;
 14   ( ROUTINE TO INITIALIZE DISK )
 15
```

```
SCR #116
  0 ( Diamond Draw- An Origional FORTH program by J. Volk 4/2/84 )
  1 : AT GOTOXY ; : INSTR CLS 1 1 AT ." Diamond Draw " 1 3 AT ." by
  2 John J. VOLK" 1 7 AT ." USE E,S,D,X,W,R,Z,C TO MOVE DIAMOND" 1 9
  3  AT ." 'Q' TO CHANGE DIAMOND COLOR" 1 11 AT ." 'O' FOR DRAW ON--
  4 'F' FOR DRAW OFF" 1 13 AT ." '.' TO CHANGE BACKGROUND COLOR" 1 2
  5 0 AT ." ---------- HAVE FUN ! ----------" 1 23 AT ." ANY KEY TO
  6 START" BEGIN ?KEY 0 > UNTIL ; 1 VARIABLE STAT  2 VARIABLE SCOLR
  7 124 VARIABLE YPOS 1 VARIABLE SCRCOLOR 94 VARIABLE XPOS HEX : SET
  8 UP GRAPHICS2  3800 ' SATR ! 2000 SSDT 20 DCOLOR ! 1028 4482 8244
  9  2810 0 SPCHAR YPOS @ XPOS @ SCOLR @ 0 0 SPRITE ;
 10 : DELAY 500 0 DO I DROP LOOP ;
 11 : STATEON 1 STAT ! ; : STATEOFF 0 STAT ! ;
 12 : SDCHANGE 10 DCOLOR +! DCOLOR @ F0 > IF 00 DCOLOR ! ENDIF 1 SCO
 13 LR +! SCOLR @ F > IF 0 SCOLR ! ENDIF SCOLR @ 0 SPRCOL DELAY ;
 14 DECIMAL : BCHANGE 1 SCRCOLOR +! SCRCOLOR @ 15 > IF 0 SCRCOLOR !
 15 ENDIF SCRCOLOR @ SCREEN DELAY ; -->

SCR #117
  0 ( Diamond Draw- Screen 2 ) DECIMAL
  1 : CHECK XPOS @ DUP 0 < IF 1 XPOS ! ENDIF 180 > IF 180 XPOS ! END
  2 IF YPOS @ DUP 0 < IF 1 YPOS ! ENDIF 250 > IF 250 YPOS ! ENDIF ;
  3 : UP -1 XPOS +! CHECK YPOS @ XPOS @ 0 SPRPUT STAT @ 1 = IF YPOS
  4 @ 3 + XPOS @ 8 + DOT ENDIF ;
  5 : DOWN 1 XPOS +! CHECK YPOS @ XPOS @ 0 SPRPUT STAT @ 1 = IF YPOS
  6  @ 3 + XPOS @ 3 + DOT ENDIF ;
  7 : RIGHT 1 YPOS +! CHECK YPOS @ XPOS @ 0 SPRPUT STAT @ 1 = IF YPO
  8 S @ XPOS @ 3 + DOT ENDIF ;
  9 : LEFT -1 YPOS +! CHECK YPOS @ XPOS @ 0 SPRPUT STAT @ 1 = IF YPO
 10 S @ 3 + XPOS @ 3 + DOT ENDIF ;
 11 : LUP -1 XPOS +! -1 YPOS +! CHECK YPOS @ XPOS @ 0 SPRPUT STAT @
 12 1 = IF YPOS @ 3 + XPOS @ 8 + DOT ENDIF ;
 13 : RUP -1 XPOS +! 1 YPOS +! CHECK YPOS @ XPOS @ 0 SPRPUT STAT @ 1
 14  = IF YPOS @ 3 + XPOS @ 8 + DOT ENDIF ;
 15 -->

SCR #118
  0 ( Diamond Draw-3rd Screen)
  1 : LDOWN 1 XPOS +! -1 YPOS +! CHECK YPOS @ XPOS @ 0 SPRPUT STAT @
  2  1 = IF YPOS @ 3 + XPOS @ 8 + DOT ENDIF ;
  3 : RDOWN 1 XPOS +! 1 YPOS +! CHECK YPOS @ XPOS @ 0 SPRPUT STAT @
  4 1 = IF YPOS @ 3 + XPOS @ 8 + DOT ENDIF ;
  5 : INIT INSTR SETUP ;
  6 : KEYIN3 CASE 69 OF UP ENDOF 88 OF DOWN ENDOF 68 OF RIGHT ENDOF
  7 83 OF LEFT ENDOF ENDCASE ; : KEYIN2 CASE 87 OF LUP ENDOF 82 OF R
  8 UP ENDOF 90 OF LDOWN ENDOF 67 OF RDOWN ENDOF ENDCASE ; : KEYIN1
  9 CASE 79 OF STATEON ENDOF 70 OF STATEOFF ENDOF 46 OF BCHANGE ENDO
 10 F 81 OF SDCHANGE ENDOF ENDCASE ;
 11 : ENDALL TEXT CLS 1 1 AT ." ENTER 'FORGET AT' TO SAVE MEMORY" ;
 12 : RUN INIT BEGIN ?KEY DUP DUP DUP DUP 0 > IF KEYIN1 KEYIN2 KEYIN
 13 3 ENDIF SP! ?TERMINAL UNTIL ENDALL ;
 14
 15
```

PC NOTES

The votes are in and it looks like the TI PC column will be staying for awhile. A few people wrote in saying that they had no intentions of buying a TI PC and as such they would like to see this space devoted to 99/4A articles. But quite a few other people said they like the column a they want it to stay so here it is for May. By the way, most of the articles that we have been placing in this column are fairly standard enough to be used with most MS-DOS based computers, such as the IBM, Compact, Eagle etc.

This month we will cover some of the new aspects of MS-DOS V2.11. This new version has some very nice enhancements added to it. The first of which is the TREE structured directories. These are most useful with a hard disk since you can easily segment the disk under different directories and keep all those files organized! When you are using subdirectories you should find the PATH command to be extremely useful. This command allows you to tell the system which path (directory-subdirectories) to search in for a file. On our system we have placed all of the DOS system files in the ROOT directory. Then in our AUTOEXEC.BAT file we have the command PATH \ . The backslash is the default name of the ROOT directory. Now when we are in any other directory we can easily use any of the DOS files with the current directory, such as TREE, SIZE, CHKDSK *.* etc. by just typing them in.

Another new feature that can be added to your system boot disk is the CONFIG.SYS file. This file is created by using the line editor EDLIN or by typing in:

COPY CON CONFIG.SYS

This will copy whatever you type in to the soon to be created CONFIG.SYS file. After you have every thing typed in that you want just press F6 or CTRL Z and RETURN. That will put an End Of File marker on the screen and copy whatever was typed into the CONFIG.SYS file.

With a CONFIG.SYS file on your boot disk you can set up a number of system parameters including RAMDISK(s), PRTSCRN (screen dump utility) and the number of Disk Buffers. Here is how our CONFIG.SYS file was set up:

```
E>COPY CON CONFIG.SYS
BUFFERS=9
DEVICE=RAMDISK.DEV 50
DEVICE=PRTSCRN.DEV
^Z                    (eof marker F6)
```

Now whenever we power up our system it looks into the CONFIG.SYS file and sets up nine Disk Buffers, one Ram Disk that can hold up to 44K of files, 45056 bytes, and it installs the Screen Dump utility into memory. The ram disk has an overhead of 6K, even though the manual says it has an overhead of 3K. We have looked in the memory where the ram disk resides and it doesn't appear to require the 6K overhead. It is possible that it is setting up a pointer wrong, which may be the reason for the following problem we ran into. Some Basic programs, such as TIDRAW.BAS, that use BLOAD to load Assembly language support routines into Basic's memory space, may lock up if you have a CONFIG.SYS file on your boot disk. The Basic programs that use BLOAD to load the Three Planes Graphics memory work fine since this is a completely different area of memory. The CONFIG.SYS file listed at the top of this column seems to work fine on every program we have tested. If your CONFIG.SYS file causes lock up, try changing the the size of the ram disk(s) or the number of buffers. We found that the larger the ram disk(s), the greater the the number of buffers that were needed to prevent lock up. Also an odd number of buffers seemed to help clear up the problem and set up the memory properly.

Once you have the ram disk installed it is accessed as **G:** and all of the DOS commands and system files will work on and with it, except FORMAT. So it is a great temporary storage area. It also helps to speed up execution of programs that require a lot of disk accesses such as Wordstar and dBASE II. It is also nice for files that you are going to use EDLIN or DEBUG on, and to temporarily store and edit files that are downloaded through your modem.

The Screen Dump utility works with the TI 850, TI 855 and Epson printers with graphics. After this utility is installed you can press; **SHIFT PRINT** to dump text only, **ALT PRINT** to dump graphics only, **CTRL PRINT** to dump reversed graphics, **ALT SHIFT PRINT** for text and graphics or **CTRL SHIFT PRINT** for reversed graphics and text. Have fun.

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

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

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