MILLERS GRAPHICS

JUNE 1984

$1.50

THE SMART PROGRAMMER

To start off this issue I would like to dispel a few rumors that are floating around. First off, as most you know by now, we have not gone out of business nor are we planning to. It seems that the phone company had a few problems a while back and was telling people that our phone was disconnected. I don't know what the problem was since our phone has always worked from this end. Next on the list is the CorComp CEO rumor. I have never been nor do I plan on being the Chief Executive Officer of CorComp Inc. Also, while we are on the CorComp subject, they did not buy us out and we did not buy them out! There has never been any such intention from either party. It sure is funny how these things get started and passed around. I guess that I'll have to get on the Source more often and keep an eye out for them. Enough of that stuff lets get started with the newsletter.

We would like to thank all of the User's Groups for their continued support, even though we slipped up and didn't release an issue for awhile. The Disk Controller project was a little more involved than we originally thought. We would also like to thank everyone for passing the good word around about the newsletter and for Bill Grants letter to Compute magazine about the newsletter. Its nice to know that everyone is pleased with it. We enjoy writing it and doing the research for it. As we stated in the May newsletter, our main goal is to get it out on a regular monthly basis. It is almost impossible to get caught up (Oct. issue out in Oct.) without slopping together a bunch of issues. So if you don't mind the June issue arriving in Sept. we would rather keep the quality up and release them monthly.

OOPS!

Other than a few misspelled words the 4/A articles were fine. In the TI PC column we slipped up on the information about MS-DOS 2.1 and the TIDRAW program. After a little closer look we discovered that the problem was where the DEF SEG was set at in the TIDRAW program and not with DOS. The original DEF SEG was at hex 1E00. This was fine under DOS 1.25 but with DOS 2.1, the RAM Disk, Buffers and the PRTSCRN Utility installed it was writing over an important part of memory. We change the DEF SEG at the beginning of the TIDRAW program to DEF SEG=&H3A00 and everything now works fine. In order for this to work you must have a minimum of 256K of RAM installed.

Q&A

We have been receiving a number of questions on console lock up (without using CALL LOAD or CALL PEEK). Many have stated that after half an hour or more of use the console just locks up when they are typing in a program or running a program that worked fine before. They also stated that sometimes the console must cool down before it will come back and other times it will come right back after they have repowered up.

There are quite a few reasons why your console my lock up so lets start with the easiest ones to fix and then move on to the hard ones. First off ANY connection is a weak link and over time they can corrode and lose contact. Also when the console warms up these connections expand slightly and may lose contact because of a little corrosion. Removing and inserting your Extended Basic module a few times will clean up the contacts on the module and the cartridge port and may clear up the problem. The side port can be cleaned with the same technique. We have also found that the Speech Synthesizer seems to add to the lock up problem since it adds another connection between your console and your Expansion Box. You might try using your computer without it to see if that helps. Next check the cartridge port connections by opening the door and looking in there with a flash light for any bent, lose or misaligned contacts in the connector. You should also check the Speech connector, the Expansion Box connector and ALL of the connectors inside the Expansion Box. Lastly check the circuit board edges (where the cards and modules plug into the connectors) for torn or rolled back contacts. Note: the female connectors have a contact in every position but the circuit boards do not always have a matching contact. Some cards and modules only use part of the possible contacts and some of them even have a number of contacts connected together.

If the above items don't help the lock up problem it may be inside your computer. This is where it gets a little harder to fix. If you have never had your console apart then I recommend that you get together with a friend that has (and got it back together OK) for the following items. I might also add that you should ground yourself to a metal object before handling the main circuit board or any chips! Static electricity can build up in your body and transfer itself to the chip. If this happens its good bye chip!! The first thing to check is the right angle connector that the modules plug into. This connector plugs down into the main circuit board and it may be dirty or have a bent contact so look it over carefully.

On the Black and Silver consoles there are a few socketed chips that may have partially walked themselves out of their sockets. So if the right angle module connector was OK on the Black and Silver console then remove the metal shielding from the main circuit board. After the shielding is off you will see a chip in the upper left hand quadrant of the board that has white silicon heat dispersing grease on it. This grease makes contact with the metal shielding to help keep this chip cool. This is the TMS 9918A Video Display Processor and it is a socketed chip. Use the eraser end of a pencil to gently push this chip back into its socket. It would not be a bad idea to apply a small amount of white silicon grease to the top of this chip to ensure good heat dispersion. This heat dispersing grease can be obtained from a good electronic store.

The other three socketed chips are GROMs 0, 1 and 2. They are located on the left of the right angle module connector above the TMS 9900 microprocessor. Once again use your pencil eraser to gently push these chips back into their sockets. If any of the chips were lose you might hear a slight click sound as you reseat them. While you have your console apart this far you might want to check all of the wire and ribbon cable connections. It might not be a bad idea to spray your key contacts with a good quality tuner/contact cleaner. This will eliminate most or all of the unwanted repeating when you press a key.

If none of the above items helped your lock up problem or keyboard repeat and you are handy with a soldering iron you MIGHT try replacing the 9901 chip in the console. It is located to the right of the TMS 9918A. This chip handles the I/O for your computer and allows it to talk to the VDP, keyborad and Cassette I/O items. This chip costs around 15.00 retail and you can probably find it for less through a discount electronics store or mail order house. I recommend that you also buy a socket for it and solder that to the board instead of the chip. Most chips are very sensitive to heat and with a socket you do not have to worry, also its a lot easier to replace this chip latter on if it goes bad again.

While we are on the subject of repair lets discuss the Expansion Box and cards. The easiest way to diagnose a problem with your P-Box is through the process of elimination. If your computer locks up on Power Up try unplugging the P-Box from the console. If it works OK then the problem is in the P-Box. If there is a problem with the P-Box start by unplugging every card but the Flex Cable Interface and try powering up your system. Then one by one plug your cards back in until the problem reoccurs. Then unplug all of the cards but the one that seems to cause the problem and the Flex Cable. If the problem has gone away then it is probably a combination of two or three cards that causes it. When you think you have it narrowed down get together with a friend that has a P-Box and try swapping cards around. The first card to swap should be the Flex Cable Interface. This card can partially go bad, which may make you think that your RS232 or Disk Controller is bad. If everything in your box works fine with your friends Flex Cable then very carefully check all of the Flex Cable connections and make sure they are not bent or misaligned. If the connections look OK the problem may be with one of the 5 chips in the Flex Cable circuitry.

On the Flex Cable Card itself there are 4 chips. Three 74LS244's and one 74LS245. Inside the black connector that plugs into the computer there is also one 74LS245 chip. These chips cost under 4.00 at a discount electronics store but they may be hard to find from time to time. Usually it is one of the 74LS244's that goes bad. The 74LS245's rarely go out. If your system no longer recognizes your RS232 or Disk Controller but they work fine in your friends system, then odds are you have a bad 74LS244. The chips on the card are lined up in a row and the 74LS244 in the middle is the most common cause of problems. If you decide to repair the card yourself then once again I recommend that you buy some 20 pin sockets for the chips and solder those to the board.

We have also discovered that in some of the Expansion Boxes there is a hidden fuse. If you can't get any power to your P-Box and the fuse in the back of the box is OK and the power cord is OK then the problem might be with the internal fuse. This fuse is a real bear to get to. You must disassemble your P-Box to point where you can easily get to the transformer. Inside the transformer, yes I said inside, is another fuse. It is located near the bottom, usually on the side that faces the card connectors, inside the plastic case. You will have to break off part of the case to get to it. It is not in a fuse holder it is just soldered in between a couple of wires. This fuse rarely goes out unless there is something else wrong with the power supply inside the P-Box. So just replacing it may not solve the problem and it may just blow out again.

We've been receiving a number of questions on upgrading the memory in the console.

After checking with a few people the conclusion is that it would be extremely difficult to modify the the RAM in the console. Replacing the 4116 RAM chips with 4164 RAM chips will not accomplish anything without a major modification to accommodate some sort of multiplexing scheme to address the additional memory. This requires a major modification to the circuitry. The biggest problem arises from the fact that TI did not use a DRAM controller to address the memory. There might also be a problem with the voltage since the 4164s draw more power. Oh well, nice thought.

We've received a number of questions on the problems with TI Writer and the formatter. The answer for most of the bugs and other problems has been taken care of by the release of a new version of TI Writer. TI recently released a newer version of TI Writer and TI Multiplan to the User's Groups. So if you belong to a group you can get these new versions. TI also released the source code for TI Forth to the User's Groups so get in contact with your local group to get copies of these programs.

We are still receiving a few questions regarding problems with the BLOAD changes to the Forth System disk.

Some people wrote in saying that everything worked fine but they were getting) ? on the screen after it loaded. We looked at the printouts of Screen 3 they sent us and on line 13 there was a space missing between ;(it should be ; (. Some of the other problems have come from modifications to our screen 3. We chose screen 51 for the BSAVE because it would not wipe out any important screens even if you loaded ALL the options. Also a few people forgot or accidentally erased the R->BASE on line 15 of Screen 3. A few others modified Screen 3 and did not close the REMs properly. The (character tells Forth that EVERYTHING up to the next) character is a remark and not program code. If you forget to close your remark statements Forth will not execute anything after the last (character! So on Screen 3 if one of the) is missing it may not execute R->BASE and it will lock up!

PEEKING AROUND

This month we mapped out GROM chip 0 in the console. Next month we will map out GROM chips 1 and 2 (Basic Language). Before we get into the memory map I would like to talk a little about GROM and the GPL Language.

GROM in the 99/4A is a memory mapped ROM device that is read and executed serially by the GPL Interpreter. The GPL interpreter resides in the console ROM and is written in 9900 Assembly Language. Each time GROM is accessed it automatically increments itself to its next address, that is why your console locks up when you CALL PEEK (-26624,x). When you execute this CALL PEEK from Extended Basic, GROM increments itself and it is on the wrong address for Extended Basic to continue execution. The GPL interpreter, or your assembly program, can change the GROM address to be read by writing to >9C02 (-25598), this allows the GPL interpreter to branch to and/or CALL different GROM routines.

GROM chips, TMS0430, contain up to 6K of data but in the 99/4A they reside in 8K address spaces. So the valid GROM addresses are >0000->17FF, >2000->37FF, >4000-57FF etc. The data that is located between >1800 and >1FFF etc. is just a repeat of some other portion of the GROM chip. The easiest way to look through GROM is to use the DEBUGGER that comes with the Editor/Assembler. After loading DSK1.DEBUG through the Editor/Assembler Load and Run option and starting it with the start name of DEBUG, press M for memory dump. Then place a 'G' before the first memory dump address (ie: M G0000,17FF) to dump the contents of GROM 0 to the screen.

If you want to look at Extended Basic's GROM you can re-Assemble the DEBUGGER without the 'C' option to a file named 'XBDEBUG' and load it with CALL INIT :: CALL LOAD("DSK1.XBDEBUG"). Note: The new uncompressed object code for the debugger requires 66 sectors on the disk. After it is loaded type in CALL LINK("DEBUG") and then press 'U' to set the proper character bias for Extended Basic. When you have the dot prompt on the screen type in:

M G6000,D7FF

to look through XB's GROM. While it is dumping the GROM data to the screen you can press the space bar to stop and start the scrolling. For more information on the Debugger see pages 363-392 in the Editor/Assembler manual. If you belong to a User's Group you can get a copy of SBUG (Super Debugger) which TI recently released to them. This has provisions for generating a hardcopy print out of memory dumps and disassembled 9900 ROM and RAM object code. It also allows you to single step through your program.

The 99/4A is set up to address 3 GROM chips in the console and up to 5 more GROM chips in a cartridge for a total of 48K. However, TI set up the 4A to allow addressing of multiple pages of cartridge GROMs. What this means is that with the proper hardware you could have up to 16 cartridges plugged in and the console could address them one at a time. However one module can access routines from another module with the GPL CALL routine. Imagine 16 times 30K + 18K (console) or 498K of GROM data on line for your use, that is a lot of power! In the GROM chip 0 map you will notice a reference to a data statement that says 'REVIEW MODULE LIBRARY'. This statement is part of a routine that searches through all 16 pages, if the proper hardware is there, and builds an application program menu for them when the Power Up routine is executed. I think TI was thinking about building an extended module holder at one time. I've been told that there is a limited amount of power available out of the cartridge port so a holder of this type would require its own power supply and some buffering on the extension cable into the port. I hope that one of the many third party hardware companies out there give it a try, I'm tired of swapping modules. If this device were a reality the memory mapping addresses for the different pages are as follows:

GROM PAGE	READ DATA	READ ADDRESS	WRITE DATA	WRITE ADDRESS
0	9800	9802	9C00	9C02
1	9804	9806	9C04	9C06
2	9808	980A	9C08	9C0A
3	980C	980E	9C0C	9C0E
4	9810	9812	9C10	9C12
5	9814	9816	9C14	9C16
6	9818	981A	9C18	9C1A
7	981C	981E	9C1C	9C1E
8	9820	9822	9C20	9C22
9	9824	9826	9C24	9C26

GROM PAGE	READ DATA	READ ADDRESS	WRITE DATA	WRITE ADDRESS
10	9828	982A	9C28	9C2A
11	982C	982E	9C2C	9C2E
12	9830	9832	9C30	9C32
13	9834	9836	9C34	9C36
14	9838	983A	9C38	9C3A
15	983C	983E	9C3C	9C3E

GPL, Graphics Programming Language, was developed by Texas Instruments. It is very compact code object code and the source code is similar to assembly but contains some higher level instructions than 9900 assembly code. This language is byte oriented instead of word oriented like 9900 assembly. The instructions can be either byte or word values and can directly access all of CPU ROM and RAM, GROM and VDP RAM. It allows data transfers from a single byte to an entire block in one instruction, from anywhere to anywhere. It also allows a special formatted data move for easily placing compacted data in exact locations on the screen. GPL also supports two stacks in CPU scratch pad RAM know as the GPL Data stack and the GPL Subroutine stack. The data stack is used to manipulate data that is pushed and popped to and from it. The subroutine stack keeps track of the subroutine return addresses to allow nesting of multiple subroutines.

GPL also contains instructions (operands) for key board scanning, random number generation, sprite coincidence detection, pattern changes, sound, I/O control and XML for linking to 9900 Assembly language routines. We understand that the GPL Assembler was written in a combination of FORTRAN and 9900 Assembly, but its only available for TI 990 minicomputer systems. Some of the higher level macros provide for IF - THEN - ELSE, REPEAT - UNTIL, CASE, MOVE (blocks of data), WHILE - END, FOR - TO - BY - END, GOTO and CALL. Lets look at a couple of GPL instructions from the Power Up routine and their assembled opcodes.

Address **Opcode** **Instruction**

>006E 390008000451

MOVE 8 FROM ROM(#VDPREG$) TO VDP(0)

This instruction moves 8 bytes from the GROM #VDPREG data table at >451 into the VDP registers (initializes the VDP registers with the default values for the Power Up routine).

>00F5 310200A90004B4

MOVE 512 FROM ROM (#CHR1$) TO CHAR(>20)

This instruction moves 512 bytes from the GROM #CHR1$ data table at >04B4 into the pattern definition table, starting with character 32 (>20). This loads the Title Screen characters.

>00C2 8EA00040DC

IF RAM(0) .NE. 0 GOTO label

This instruction checks the byte at VDP RAM location >00 and IF it is Not Equal to 0 it branches the location pointed to by the label relative to GROM chip 0 (>40 = BRANCH to location >00DC).

>01B4 0603CE CALL BEEPTONE

This instruction CALL's (GOSUB's) the routine to generate a beep sound located at >03CE. When a >00 (RTN) or >01 RTNC is encountered in the CALLed routine it returns back to where it was called from and continues execution at the next instruction.

The following GROM memory maps are from a console that has the coding of LTA 0683. This LTA number indicates the week and year the console was assembled. You may find when you are looking through your GROM with the Debugger that it does not exactly match the memory maps on the following pages. TI made a number of small changes to both the console ROM and GROM over the years so not every console is exactly the same. These changes are the main reason you should always use the vector tables to acquire the entry points for the different routines instead of branching directly into the routine with your assembly language programs. If you branch directly into the routine on your console it may not work on your friends consoles. Unfortunately TI has not released a complete list of the changes and their related LTA or ATA dates for the approximately 6 different versions of consoles.

The GROM routines that are listed in the GROM vector table can be called from an Assembly Language, Forth or Extended Basic program (with EXEGPL from the May issue). The Editor/Assembler manual gives a fairly complete description of the proper setup for calling each routine on pages 251-257. I hope this will help in your understanding of GROM and its possible applications in your Assembly, Forth and XB programs.

CONSOLE GROM CHIP 0

```
+-----------------------------------------------------------------------+
|>0000   | GROM HEADER                                                  |
|  >0000 | >AA   Valid GROM Header Identification Code                  |
|  >0001 | >02   Version number                                         |
|  >0002 | >0000 Number of Programs.                  ....  none here    |
|  >0004 | >0000 Address of Power Up Header           ....  none here    |
|  >0006 | >0000 Address of Application Program Header ....  none here   |
|  >0008 | >1310 Address of DSR Routine Header                          |
|  >000A | >1320 Address of Subprogram Header                           |
|  >000C | >0000 Address of Interrupt Link             ....  none in GROM|
|  >000E | >0000 Reserved for future? expansion.                        |
|        |                                                              |
|>0010   | GPLLNK VECTOR TABLE                                          |
|        | The values in these tables contain the instruction >40 (BR)  |
|        | which is BRANCH if condition bit in status register is RESET |
|        | and the address is relative to the 6K GROM chip it resides in.|
|        | Actual address for GROM 0 = value - >4000 (ie: >43DC = >03DC)|
|        |                                                              |
|  >0010 | >43DC LINK programs to link between programs and DSR's       |
|  >0012 | >443C Return from LINK or DSR                                |
|  >0014 | >49A9 CNS - Convert number into a string                     |
|  >0016 | >4396 Load Title screen characters                           |
|  >0018 | >439E Load Regular upper case characters                     |
|  >001A | >4446 Generate Basic WARNING message                         |
|  >001C | >4449 Generate Basic ERROR message                           |
|  >001E | >444C Begin execution of GROM Basic                          |
|  >0020 | >4052 GROM Power Up routine                                  |
|  >0022 | >51FE INT - Convert floating point to Integer function       |
|  >0024 | >4C82  ^  - Exponentiation, raise a number to a power        |
|  >0026 | >4D59 SQR - Square Root function                             |
|  >0028 | >4DB4 EXP - Exponential function                             |
|  >002A | >4E64 LOG - Natural Logarithm function                       |
|  >002C | >4EF9 COS - Cosine function                                  |
|  >002E | >4F01 SIN - Sine function                                    |
|  >0030 | >4F5F TAN - Tangent function                                 |
|  >0032 | >4F80 ATN - Arctangent function                              |
|  >0034 | >43CE     - Generate BEEP sound                              |
|  >0036 | >43D6     - Generate HONK sound                              |
|        |                                                              |
|  >0038 | >054D12 = BRANCH to GROM 2  >4D12 Get String Space routine   |
|  >003B | >525E     - Bit reversal routine                             |
|  >003D | >4417     - Special GROM entry point for Cassette DSR, points to|
|        |             a GROM routine that calls an XML to execute the low |
|        |             level Cassette DSR in the console ROM which returns |
|        |             to the high level Cassette DSR in GROM.             |
|  >003F | >052844 = BRANCH to GROM 1  >2844 Memory space check for PAB's |
|  >0042 | >0537B4 = BRANCH to GROM 1  >37B4 GPL subprogram setup       |
|  >0045 | >60     = DATA - Basics screen character offset              |
|  >0046 | >0D00   = DATA - Speech Read address  (>0D00 + >8300 = >9000)|
|  >0048 | >1100   = DATA - Speech Write address (>1100 + >8300 = >9400)|
|        |                                                              |
|  >004A | >43C2 Load Lower case characters                             |
|        | --- The following three were changed in the later version of --- |
|        | ---             GROM - After approx 3/82 or LTA 1482       --- |
|  >004C | >04B4 Address of the Title Screen character data table       |
|  >004E | >06B4 Address of the Regular upper case character data table |
|  >0050 | >0874 Address of the Lower case character data table         |
+-----------------------------------------------------------------------+
```

CONSOLE GROM CHIP 0 Cont.

	GROM ROUTINES
>0052	POWER UP ROUTINE (displays the Title Screen see MAY newsletter)
>0396	LOAD TITLE SCREEN characters routine
>039E	LOAD REGULAR UPPER CASE characters routine
>03C2	LOAD LOWER CASE characters routine
>03CE	GENERATE BEEP sound routine
>03D6	GENERATE HONK sound routine
>03DC	LINK ROUTINES for linking between programs and DSR's
>043C	RETURN from link or DSR
>0446	>05284C = BRANCH to GROM 1 >284C - WARNING routine
>0449	>05284E = BRANCH to GROM 1 >284E - ERROR routine
>044C	>052010 = BRANCH to GROM 1 >2010 - Execute Basic
>044F	**DATA TABLES**
>044F	DATA >80 (Hex 80)
>0451	DATA for VDP Register default values
>0459	DATA for Color Table default values for Title Screen
>0479	DATA for BEEP sound
>0484	DATA for HONK sound
>048F	DATA :1981:
>0496	DATA :TEXAS INSTRUMENTS:
>04A7	DATA :HOME COMPUTER:
>04B4	DATA for Title Screen Characters (CHR$(32-95))
>06B4	DATA for Regular Upper Case Characters (CHR$(32-95))
>0874	DATA for Lower Case Characters (CHR$(96-126))
>094D	DATA :FOR:
>0950	DATA for TI LOGO loaded at CHR$(1) in the Pattern Desc. Table
>09A0	**FLOATING POINT ROUTINES**
>09A0	Roll Out routine- moves part of Scratch Pad to VDP Roll Out Area
>09A9	CNS - Convert Number into String routines
>0AE6	Roll In routine- moves VDP Roll Out Area back into Scratch Pad
>0AEF	Balance of CNS routines
>0C6C	V PUSH - Push a number from FAC onto the VDP Value Stack
>0C77	V POP - Pop a number off the VDP value Stack to FAC
>0C82	^ - Exponentiation, raise a number to a power
>0D59	SQR - Square Root function
>0DB4	EXP - Exponential function
>0E64	LOG - Natural Logarithm function
>0EF9	COS - Cosine function
>0F01	SIN - Sine function
>0F5F	TAN - Tangent function
>0F80	ATN - Arctangent function
>0FDB	DATA and misc constants used by the Floating Point routines
>117B	Misc subroutines used by the Floating Point routines
>11FE	INT - Integer function
>125E	BIT REVERSAL ROUTINE (see May newsletter)
>1267	DATA this is the >40 bytes that is moved into >8300 and used by the Bit reversal routine.
>12A5	DATA :REVIEW MODULE LIBRARY: (currently not used - this is for the Multi-Page GROM set up, called from Power Up >01FC)
>12C0	/ \
>130F	\ Unused area contains >0000 /

CONSOLE GROM CHIP 0 Cont.

	CASSETTE DSR - High Level - checks for OPEN errors, displays screen messages for cassette operation etc.
	PAB set up for DSR (see Editor/Assembler manual pages 291-304)
	PAB+0 - I/O Opcode (Open, Close, Load, Save etc.)
	PAB+1 - Flag/Status (File-type, Mode of Operation & Data-type)
	PAB+2 - VDP Data Buffer Address
	PAB+4 - Logical Record Length
	PAB+5 - Character Count (bytes) to be transferred
	PAB+6 - Record Number (0-32767 not used for cassette I/O)
	PAB+8 - Bias for ASCII characters (>60 in Basics)
	PAB+9 - Length of the Device Name (>03 for CS1)
	PAB+10 - Start of the Device Name 'CS1' or 'CS2'
>1310	**DSR Header(s)**
>1310	>1318 - Pointer to next Device Name Header
>1312	>1326 - Start address for this Device
>1314	>03 - Name length for this Device
>1315	>435331 - DATA :CS1:
>1318	>0000 - Pointer to next Device Name Header - no more
>131A	>132C - Start address for this Device
>131C	>03 - Name length for this device
>131D	>435332 - DATA :CS2:
>1320	**SUBPROGRAM Header**
>1320	>0000 - Pointer to next Subprogram header - no more
>1322	>1573 - Start address for this Subprogram
>1324	>01 - Name length for this subprogram
>1325	>03 - DATA :03: (can not CALL CTRL C (CHR$(3)) from Basics)
>1326	Start of CS1 DSR (set up for CS1)
>131A	Start of CS2 DSR (set up for CS2)
>1330	Both CS1 & CS2 come here to start DSR
>1374	DO CASE Branch table for OPEN, CLOSE, READ Record, WRITE Record, RESTORE/REWIND, LOAD, SAVE, DELETE(close)
>135D	ERROR and EXIT routines
>1387	**CASSETTE ROUTINES**
>1387	OPEN a file routine
>13CF	READ a Record routine
>13DA	WRITE a Record routine
>13DD	Transfer data routine for READ and WRITE
>13F2	LOAD a file routine
>140E	CLOSE a file routine
>1444	VERIFY cassette data routine
>1489	SAVE a file routine
>1499	**CASSETTE SUBROUTINES** - These subroutines display the messages on the screen for cassette operation, turn on/off the cassette motors, look for key presses and wait for the leader to pass.
>1549	Cassette Motor on
>155E	Cassette Motor off
>1562	Wait for leader to pass
>1573	SUBPROGRAM >03 - Adds Bias >60 to the Cassette messages
>15A0	**DATA TABLE** for Cassette operation messages ie: REWIND CASSETTE TAPE, THEN PRESS ENTER etc.
>16DC	/ I believe this area if full of junk, not used,\
>17FF	\ there are no references or branches into here./ CRC data

ASSEMBLY LANGUAGE TO EXTENDED BASIC CALL LOADS CONVERSION PROGRAM

Paolo Bagnaresi of Milan, Italy sent us the nice conversion program on the following pages. Here are a few excerpts from his letter that help to explain the program. "The program of Barry Traver is really interesting. I followed your advice of using this MERGE approach as a new tool of programming, as well as your tips and explanations about memory locations. I developed this program called ACE (Assembly Converted to Extended), which by means of CALL PEEKs, peeks into low memory expansion. ACE then creates a MERGEable DIS/VAR 163 Extended Basic file that contains the actual content of the memory locations in CALL LOAD form. ACE also converts the DEF table, the pointer to the DEF table and the pointer to the First Free Address in low memory." "The original Assembly program must RUN by itself in the Extended Basic environment and it must not contain any AORG statements." "To use ACE load the program (making sure that the low memory expansion is clean, which is, start from the master title screen). The program is self explanatory. Just follow the instructions until the end of the program." We have used this program to convert a number of Assembly programs and it works extremely well! - Thank You Paolo -

```
1 ! ACE : Assembly Object to
   Extended Basic CALL LOADs
    Converter       8/3/1984

2 !             By
       Paolo Bagnaresi
      Tel.(02)-514.202
Address:

3 !  Via J.F. Kennedy 17
  20097 San Donato Milanese
       (Milan)- Italy

10 GOTO 40 :: DIM STDEF$(100
):: D$,N$,F$,DEF$,PB$,SC$,RI
$,CT$,L$,HEX$,H$,DSC$,DECC$,
PROG$

20 CALL LOAD :: CALL INIT ::
 CALL LINK :: CALL PEEK :: C
ALL CHAR :: CALL HCHAR :: CA
LL KEY

30 AUT,N,A,B,C,D,E,F,G,H,I,L
,M,N,CT,MS,LS,DBM,DBL,FINELO
C,LOC,INIZLOC,INDEF,NDEF,NLI
NK,NL,NLINE,NST,KY,ST,DEC,PO
,Z

40 CALL CLEAR :: CALL SCREEN
(16):: FOR T=0 TO 14 :: CALL
 COLOR(T,13,16):: NEXT T ::
!@P-

50 CALL CHAR(128,"00282828",
131,"000000FF"):: L$=RPT$(CH
R$(131),28):: H$="0123456789
ABCDEF" :: CALL CLEAR

60 PB$="By Paolo Bagnaresi
           Via J.F. Kennedy
17         20097 San Donato
Milanese   (Milan)- ITALY"

70 DISPLAY AT(1,1):L$:L$: :T
AB(13);"ACE": :TAB(14);"by":
TAB(7);"Paolo Bagnaresi":TAB
(7);"Tel(02)-514.202":"San D
onato Milanese-ITALY":L$

80 DISPLAY AT(11,1):TAB(10);
"Assembly":TAB(10);"Converte
r to":TAB(10);"Extended":L$

90 DISPLAY AT(14,1):L$:"ACE
converts the Object":"of an
Assembly Program into":"an E
xtended Basic Program.":"The
 Assembly Program MUST be"

100 DISPLAY AT(19,1):"suitab
le for Extended Basic":"envi
ronment and MUST NOT":"conta
in any AORG.":L$

110 DISPLAY AT(24,6)BEEP:"Pr
ess any key"

120 CALL KEY(0,KY,ST):: IF S
T=0 THEN 120 ELSE CALL CLEAR

130 DISPLAY AT(1,1):L$:TAB(5
);"Are the Assembly":"Object
s already loaded ?": :"  Ans
wer: (Y/N) N":L$

140 ACCEPT AT(5,17)SIZE(-1)V
ALIDATE("YN")BEEP:RI$ :: IF
RI$="" THEN 50 ELSE IF RI$="
Y" THEN CALL CLEAR :: GOTO 3
70 ELSE N$="1" :: CT$="Y" ::
 GOSUB 900
```

```
150 DISPLAY AT(6,1):L$:"Inse
rt the diskette with the":"a
ssembly object and enter":"t
he object progr. name": :"Na
me :  ";PROG$:L$

160 DISPLAY AT(22,1):L$:"era
se and press ENTER if you":T
AB(9);"are through"

170 ACCEPT AT(11,9)SIZE(-10)
BEEP:PROG$ :: IF PROG$="" TH
EN CALL CLEAR :: GOTO 320

180 DISPLAY AT(13,1):"Disk D
rive ? (1-3) ";N$:L$ :: ACCE
PT AT(13,20)SIZE(-1)VALIDATE
("123")BEEP:N$ :: IF N$="" T
HEN 130

190 ON ERROR 360 :: CALL LOA
D("DSK"&N$&"."&PROG$):: ON E
RROR STOP

200 DISPLAY AT(15,1):"Do you
 want to check the":"loaded
program ? (Y/N) ";CT$:L$ ::
ACCEPT AT(16,24)SIZE(-1)VALI
DATE("YN")BEEP:CT$ :: IF CT$
="N" THEN 150

210 DISPLAY AT(18,1):"Does t
he program come back":"to Ex
tended Basic ? (Y/N) Y":L$ :
: ACCEPT AT(19,27)SIZE(-1)VA
LIDATE("YN")BEEP:SC$

220 IF SC$="Y" THEN 240 ELSE
 FOR T=1 TO 10 :: DISPLAY AT
(21,1)BEEP:"In this case no
check":"is possible ":L$:L$

230 FOR I=1 TO 100 :: NEXT I
 :: CALL HCHAR(21,1,32,96)::
 NEXT T :: GOTO 150

240 CALL PEEK(8196,A,B):: NS
T=0 :: INDEF=A*256+B :: FOR
T=16376 TO INDEF STEP -8 ::
NL=T :: GOSUB 870 :: NST=NST
+1 :: STDEF$(NST)=DEF$ :: NE
XT T

250 CALL CLEAR :: A=0 :: D$=
"1"

260 DISPLAY AT(1,1):L$:"List
 of DEFS to choose from":"fo
r checking pourposes":L$270
FOR T=5 TO 20 STEP 2 :: FOR
Z=1 TO 19 STEP 9 :: A=A+1 ::
 DISPLAY AT(T,Z)BEEP:A;STDEF
$(A):: IF A>=NST THEN 280 EL
SE NEXT Z :: NEXT T

280 DISPLAY AT(T+1,1):L$ ::
DISPLAY AT(20,1):L$:"DEF No.
 ? (1 -";NST;") ":L$:"Press
ENTER when finished":L$ :: A
CCEPT AT(21,21)VALIDATE(DIGI
T)SIZE(-2)BEEP:D$ :: IF D$<>
"" THEN 340

290 CALL CLEAR :: DISPLAY AT
(1,1):L$:"Are the programs l
oaded": :"so far OK. ? (Y/N)
 Y":L$ :: ACCEPT AT(4,20)VAL
IDATE("YN")SIZE(-1)BEEP:SC$
:: IF SC$="Y" THEN 320

300 DISPLAY AT(6,1):L$:"Unfo
rtunately in this case": :"i
t's not possible to": :"elim
inate just one program": :"b
ut it's necessary to load"

310 DISPLAY AT(15,1):"all th
e program(s) all ": :"over a
gain.": :"OK? (Y) Y":L$ :: A
CCEPT AT(19,9)VALIDATE("Y")S
IZE(-1)BEEP:SC$ :: CALL INIT
 :: CALL CLEAR :: GOTO 150
320 DISPLAY AT(6,1):L$:"Are
all the programs": :"loaded
already ? (Y/N) Y":L$ :: ACC
EPT AT(9,24)VALIDATE("YN")SI
ZE(-1)BEEP:SC$

330 IF SC$="N" OR SC$="" THE
N CALL CLEAR :: GOTO 150 ELS
E 370

340 A=VAL(D$):: IF A>NST THE
N 280 ELSE CALL LINK(STDEF$(
A)):: GOTO 250

350 ! Error handling

360 FOR T=1 TO 8 :: DISPLAY
AT(20,1)BEEP:L$:TAB(6);"Driv
e error or":TAB(6);"Program
name error":L$:L$ :: FOR I=1
 TO 100 :: NEXT I :: CALL HC
HAR(20,1,32,128):: NEXT T ::
 RETURN 150
```

```
370 CALL CLEAR

380 ON ERROR 400 :: CALL PEE
K(8194,A,B,C,D):: FINELOC=A*
256+B :: NL,INDEF=C*256+D ::
 GOSUB 870 :: IF ASC(DEF$)=2
55 THEN 400 ELSE INIZLOC=DBM
*256+DBL

390 ON ERROR STOP :: GOTO 43
0

400 CALL CLEAR :: FOR I=1 TO
 10 :: DISPLAY AT(10,1)BEEP:
L$:L$:"  The Assembly Progra
ms": :"   have not been load
ed": :TAB(10);"LOAD THEM!":L
$:L$

410 FOR T=1 TO 100 :: NEXT T
 :: CALL HCHAR(12,1,32,160):
: NEXT I :: GOSUB 900 :: GOT
O 150

420 !Disk-printing routine

430 CALL CLEAR :: GOSUB 790
:: IF F$="" OR N$="" THEN 32
767 :: ON ERROR 840 :: GOSUB
 920 :: OPEN #2:"DSK"&N$&"."
&F$,VARIABLE 163 :: ON ERROR
 STOP :: N=0

440 !Address of the programm
er

450 PRINT #2:CHR$(0)&CHR$(N)
&CHR$(131)&CHR$(199)&CHR$(LE
N(PB$))&PB$&CHR$(0):: N=1 ::
 GOSUB 940

460 !Insert CALL INIT

470 PRINT #2:CHR$(0)&CHR$(N)
&CHR$(157)&CHR$(200)&CHR$(4)
&"INIT"&CHR$(0):: N=2 :: LOC
=INDEF :: GOSUB 940

480 ! DEFs name printing

490 FOR NDEF=INDEF TO 16376
STEP 8

500 PRINT #2:CHR$(0)&CHR$(N)
&CHR$(157)&CHR$(200)&CHR$(4)
&"LOAD"&CHR$(183)&CHR$(200)&
CHR$(LEN(STR$(NDEF)))&STR$(N
DEF);

510 FOR LOC=NDEF TO NDEF+6 S
TEP 2

520 CALL PEEK(LOC,MS,LS):: P
RINT #2:CHR$(179)&CHR$(200)&
CHR$(LEN(STR$(MS)))&STR$(MS)
&CHR$(179)&CHR$(200)&CHR$(LE
N(STR$(LS)))&STR$(LS);

530 NEXT LOC

540 PRINT #2:CHR$(182)&CHR$(
0):: GOSUB 940 :: N=N+1 :: N
EXT NDEF

550 !Print DEF pointer and F
FALM

560 PRINT #2:CHR$(0)&CHR$(N)
&CHR$(157)&CHR$(200)&CHR$(4)
&"LOAD"&CHR$(183)&CHR$(200)&
CHR$(LEN(STR$(8194)))&STR$(8
194);

570 FOR LOC=8194 TO 8196 STE
P 2

580 CALL PEEK(LOC,MS,LS):: P
RINT #2:CHR$(179)&CHR$(200)&
CHR$(LEN(STR$(MS)))&STR$(MS)
&CHR$(179)&CHR$(200)&CHR$(LE
N(STR$(LS)))&STR$(LS);

590 NEXT LOC

600 PRINT #2:CHR$(182)&CHR$(
0):: GOSUB 940 :: N=N+1 :: L
OC=9460

610 ! Main program printing

620 PRINT #2:CHR$(0)&CHR$(N)
&CHR$(157)&CHR$(200)&CHR$(4)
&"LOAD"&CHR$(183)&CHR$(200)&
CHR$(LEN(STR$(LOC)))&STR$(LO
C);

630 FOR LOC=LOC TO LOC+20 ST
EP 2

640 IF LOC>FINELOC THEN 670
:: CALL PEEK(LOC,MS,LS):: PR
INT #2:CHR$(179)&CHR$(200)&C
HR$(LEN(STR$(MS)))&STR$(MS)&
CHR$(179)&CHR$(200)&CHR$(LEN
(STR$(LS)))&STR$(LS);

650 NEXT LOC
```

```
660 PRINT #2:CHR$(182)&CHR$(
0):: GOSUB 940 :: N=N+1 :: I
F LOC<=FINELOC THEN 620 ELSE
 680

670 PRINT #2:CHR$(182)&CHR$(
0):: GOSUB 940

680 N=N+1

690 ! CALL LINK printing

700 FOR NLINK=INDEF TO 16376
 STEP 8 :: NL=NLINK :: GOSUB
 870

710 PRINT #2:CHR$(0)&CHR$(N)
&CHR$(157)&CHR$(200)&CHR$(4)
&"LINK"&CHR$(183)&CHR$(199)&
CHR$(LEN(DEF$))&DEF$&CHR$(18
2)&CHR$(0):: GOSUB 940

720 N=N+1 :: NEXT NLINK

730 PRINT #2:CHR$(255)&CHR$(
255):: CLOSE #2

740 CALL CLEAR :: DISPLAY AT
(5,1)BEEP:L$:"The assembly p
rogram ";DEF$: :"has been re
corded as a": :"DIS/VAR 163
file. The name": :"of this f
ile is ";F$:L$

750 DISPLAY AT(14,1):"You ca
n MERGE this file": :"and ob
tain an Ext.B.Program":L$:"E
xecute now in command mode:"
: :">NEW":">MERGE DSK";N$;".
";F$

760 FOR T=1 TO 70 :: DISPLAY
 AT(23,1)BEEP:">SAVE DSK";N$
;".";SEG$(F$,1,LEN(F$)-3)&"E
XT":L$ :: CALL KEY(0,KY,ST):
: IF ST<>0 THEN STOP

770 NEXT T :: END

780 ! Open file: disk drive
& name selection

790 DISPLAY AT(1,1):L$:"Name
 of the last DEF":"of the  a
ssembly programs": :"loaded
in memory : ";DEF$:L$

800 F$=DEF$&"MRG" :: DISPLAY
 AT(8,1):L$:"proposed name f
or the file": :"Max 10 chara
cters ";F$: :L$ :: ACCEPT AT
(11,19)SIZE(-10)BEEP:F$

810 IF F$="" THEN RETURN ELS
E IF POS(F$," ",1)>0 OR POS(
F$,".",1)>0 THEN 800

820 DISPLAY AT(14,1):L$:"Dis
k Drive? (1-3) ";N$:L$ :: AC
CEPT AT(15,19)VALIDATE("123"
)SIZE(-1)BEEP:N$ :: RETURN

830 ! Sub file error

840 ON ERROR 850 :: CLOSE #2

850 RETURN 430

860 ! call peek DEF names

870 CALL PEEK(NL,E,F,G,H,I,L
,DBM,DBL):: DEF$=CHR$(E)&CHR
$(F)&CHR$(G)&CHR$(H)&CHR$(I)
&CHR$(L)

880 PO=POS(DEF$," ",1):: IF
PO>0 THEN DEF$=SEG$(DEF$,1,P
O-1):: RETURN ELSE RETURN

890 ! Sub CALL INIT once onl
y

900 IF CT=1 THEN RETURN ELSE
 CALL INIT :: CT=1 :: RETURN

910 ! Sub # of necessary pri
ntings

920 NLINE=ABS(INT(-((FINELOC
-9460)/22+(16384-INDEF)/4+3)
)):: DISPLAY AT(17,1)BEEP:"T
he necessary Printing":"oper
ations with Disk Drive": :"(
max 172) will be";NLINE:L$

930 IF NLINE>172 THEN FOR T=
1 TO 10 :: FOR I=1 TO 90 ::
NEXT I :: CALL HCHAR(23,1,32
,32):: DISPLAY AT(22,1)BEEP:
L$:"OBJECT SIZE IS TOO  LARG
E":L$ :: NEXT T :: STOP ELSE
 RETURN

940 NLINE=NLINE-1 :: DISPLAY
 AT(21,1):L$:"# of printings
 yet to be":"executed will b
e";NLINE:L$ :: RETURN
```

5^{TH} 1- =FORTH

This month we have another single disk drive disk copying routine. Right after the May newsletter went to press we received this routine in the mail from Doug Smith of Waldorf, MD. We understand that he also sent copies of this routine to a number of Users Groups around the country. In Doug's letter to us he stated that the routine destroys part of the character table and that there were a couple of other precautions to follow when using this routine. It was also mentioned that executing COLD after the routine was finished would not restore the character sets and that it was necessary to go back to the TI Title Screen and reload Forth.

The routine as we received it did a very nice job of copying a single sided single density disk in only 3 passes! Doug uses almost all of the available memory in the computer, including part of the Pattern Descriptor Table, as a buffer for the 30 screens (30K) the routine copies in one pass. The routine also assumed that there were no - (dash) options loaded (ie: -DUMP) into memory since it uses the dictionary space for a buffer also. Doug wrote the routine so that it will run with just the basic Forth system loaded.

Since most of you have modified your Forth disks to BLOAD the options you want we felt that we better modify Doug's routine to compensate for this. We also made a few other modifications to the routine to correct the Pattern Descriptor Table problem and we changed a little bit of the other logic around and added the BEEP sound for the screen prompts. Now after you have typed this program onto one of your blank screens or onto anther disk that you use for your Forth Programs just type in the screen number LOAD and press ENTER. The modified routine will clear out your BLOADed options and then load in Doug's new words. After they are loaded and complied the program displays 'ENTER W TO COPY THE DISK IN DRIVE 1 ok'. At this point you are in the command level of Forth and not in a running program so you can type in whatever you want, but remember that all of your normal options have been erased from the dictionary. This was done with the **FORGET MENU** statement on LINE 0 of the first screen.

Doug also made provisions for copying part or all of a single sided disk. At the command level you can type in **29 CY** and press ENTER to copy screens 0-29. You can also type in **59 CY** to copy screens 0-59 or just type in **W** to copy screens 0-89 (the entire disk).

The main modifications we made to Doug's routine are:
1st Screen's Lines:
0 - **BASE->R DECIMAL FORGET MENU**, this saves the current base to the return stack, changes the base to decimal and clears out any options that may be loaded in the dictionary.
3 - **52 0 33660 C! 10 SYSTEM**, this generates the beep sound when the prompts are displayed.
We also swapped lines 6 & 7, removed TX from the PR word and added the CR's, spaces and PR to line 7 (M3) to reprompt you for another disk copy.

2nd Screen's Lines:
We crunched up BPU and BDR to make room for the following additions and changes.
9 - **M2** . was added to PAS to initially prompt you for the Master disk and to allow for some logic changes in CY.
10 - **CHR** was added to restore the Pattern Descriptor table to its original state.
11 & 12 - CY was modified to accommodate the the changes made to the other lines, to make use of the word **CHR** and to display TI FORTH on the screen when the routine is finished.
14 - was added to set **DISK_LO** at **0** to avoid a DISK FENCE ERROR and **0 33730 C!** was added to allow you to use FCTN QUIT to get back to the Title Screen. MON is not in the dictionary after this routine is loaded.
15 - was added to restore the base from the return stack. You MUST have an **R->BASE** for every **BASE->R** statement or you will lock up your system!! Any time you move something out to the return stack you must at some point recall it from the return stack. And, finally, **TX** was added before PR to clear the screen and generate a beep prior to the initial prompt.

We would like to thank Doug Smith for this nice routine, we are sure that everyone with a single drive system will be pleased. The program is very simple to use and it certainly makes it a lot easier to back up your diskettes. As always I recommend that you keep the Master Diskette write protected while copying. Have Fun.

```
SCR #98
( 3 Pass Disk Copy - Doug Smith) BASE->R DECIMAL   FORGET MENU
: CLS 16 SYSTEM ; : VMBW 2 SYSTEM ; : VMBR 6 SYSTEM ;
     0 VARIABLE AREA  15360 ALLOT 0 VARIABLE PL
: TX CLS 5 10 GOTOXY 52 0 33660 C! 10 SYSTEM ;
: M1 TX ." INSERT COPY DISK-PRESS ANY KEY " KEY DROP ;
: M2 TX ." INSERT  MASTER - PRESS ANY KEY " KEY DROP ;
: PR    ." ENTER W TO COPY DISK IN DRIVE 1 " ;
: M3 TX ."       COMMAND COMPLETED " CR CR CR ."     " PR ;
: PU PL @ 20 + PL @ 5 + DO I BLOCK AREA 2 + I PL @ 5 + - 1024
     * + 1024 CMOVE LOOP ;
: BU PL @ 5 + PL @ DO I BLOCK UPDATE LOOP M1 FLUSH ;
: DR PL @ 10 + PL @ 5 + DO AREA 2+ I PL @ 5 + - 1024 * + I
     BUFFER 1024 CMOVE UPDATE LOOP FLUSH PL @ 15 + PL @ 10 + DO
     AREA 2+ I PL @ 5 + - 1024 * + I BUFFER 1024 CMOVE UPDATE
     LOOP FLUSH PL @ 20 + PL @ 15 + DO AREA 2+ I PL @ 5 + -
     1024 * + I BUFFER 1024 CMOVE UPDATE LOOP FLUSH ;   -->

SCR #99
( 3 Pass Disk Copy cont. )
: BPU PL @ 28 + PL @ 20 + DO I BLOCK 5120 I PL @ - 20 - 1024
      * + 1024 VMBW LOOP PL @ 28 + BLOCK 3072 1024 VMBW
      PL @ 29 + BLOCK 1122 1024 VMBW ;
: BDR PL @ 25 + PL @ 20 + DO 5120 I PL @ - 20 - 1024 * + I
      BUFFER 1024 VMBR UPDATE LOOP FLUSH PL @ 28 + PL @ 25 +
      DO 5120 I PL @ - 20 - 1024 * + I BUFFER 1024 VMBR UPDATE
      LOOP 3072 PL @ 28 + BUFFER 1024 VMBR UPDATE
      1122 PL @ 29 + BUFFER 1024 VMBR UPDATE FLUSH ;
: PAS M2 . BPU PU BU BDR DR ;
: CHR 2048 98 255 20 SYSTEM 3072 1024 255 20 SYSTEM ;
: CY  0 PL ! 30 / 2+ 1 DO I PAS 30 PL +! LOOP
      SP! M3 CR CR CHR ABORT ;
: W   89 CY ; ( Note: or 29 CY , 59 CY copies part of the disk)
0 DISK_LO !  0 33730 C! ( QUIT ON )
R->BASE       TX PR

SCR #100
( Instructions for 3 Pass Single Drive Disk Copy

  This program does not require any - (dash) Options to be
Loaded into memory. It will automatically clear out any that may
be loaded, since it requires almost all of the available memory.
After you are finshed copying your disks place your Forth
system disk in drive one and type in COLD and press ENTER to
reload the Forth system back into memory. To copy an entire
single sided disk type W and press ENTER, then follow the
prompts on the screen untill the command is done.

                    Given Free To Public Domain
                           Doug Smith
                         5021 Nicholas Rd.
                         Waldorf, MD 20601
                           301 645-1432
```

PC NOTES

This month we have a little Basic program that allows you to PEEK any where within the entire system memory (1 MEG). Before we get started on that I would like to let you know about a magazine for the TI PC. It is called 'TI Professional Computing' and it is published monthly by Publications & Communications, Inc.. They are located at 12416 Hymeadow Drive, Suite 2, Austin, Texas 78750-1896. Their toll free number for out of the state of Texas is 1-800-531-5093 and for Texas their number is 512-250-9023. The subscription rates for 1 year are; U.S. - $39.00 & International - $79.00. They carry a lot of nice articles, software/hardware reviews and ads for the TI PC.

Now back to Basic. After you have typed in the program at the bottom of the page save it in ASC format with the SAVE "PEEK.ASC",A command. This will allow you to MERGE it to the top of any other Basic program that starts at line number 10 with the MERGE "PEEK.ASC" command. Once it is loaded and you type RUN it asks you to input the Memory segment you want to PEEK at in Hexadecimal form. If you want to look through the Basic Data segment (Basic's program space) just press RETURN otherwise input a hex value between >0 and >FFFF. Next it asks for the starting address and the number of bytes to display, both of these inputs are in decimal form. Once these are input it takes off and starts displaying the contents of the memory you asked for. While its running you can press the BRK/PAUSE to stop it or you can press any other key to change the Memory segment or PEEK addresses. With a few simple changes you could easily OPEN LPT1: as #1 and print out the data to your printer or you can press PAUSE and SHIFT-PRINT to dump the screen to your printer. If you have a monochrome monitor you might want to change the COLOR statements in Lines 3 & 7.

Here are some Memory segments you might want to PEEK into:

>0040	**- IO SYS**
>C000	**- 3 Planes Graphics RAM Blue Plane**
>C800	**- 3 Planes Graphics RAM Red Plane**
>D000	**- 3 Planes Graphics RAM Green Plane**
>DE00	**- Active Screen Display memory**
>F800	**- Winchester Option ROM**
>FE00	**- Pegasus System ROM**

In the Basic Data segment you can find the Function Key assignments at address 1587 and your Basic program in crunched form starts around address 3457. The variable symbol table usually follows right after the crunched program.

While we are talking about Basic I would like to mention a few commands that TI forgot to tell you about. The first is **BEEP**, you can type this in direct or place it in a program and it will generate the BEEP sound. The next is **ON KEY(xx)** and **KEY(xx) ON/OFF**. This is a very handy set of commands for programming and using the function keys. To program a function key you place the command **ON KEY(#) GOSUB xxxx** in the beginning of your program. The valid values for # are 1 through 16, 1 - 12 are the function keys at the top of the keyboard and 13 - 16 are the cursor control keys, 13 = Up, 14 = Left, 15 = Right and 16 = Down. Now when ever you want to use your assignments just issue the command in your program, **KEY(#) ON** or to stop them from branching to your subroutine use **KEY(#) OFF**. You can also use a FOR NEXT LOOP like FOR I=1 to 16: KEY(I) ON: NEXT - to turn them all on or off. Once they are turned on and assigned a subroutine to branch to you do not need an INKEY$ to check for them. They are checked on interrupts, and if pressed your program automatically branches to the subroutine you specified, executes it and then returns were it left off. Another nice feature is that by turning them all on but not assigning a subroutine to them disables the INKEY$ command from checking for the function keys. Have Fun.

```
1 '<<< Memory PEEKer >>> by Craig Miller 6-04-84'
2 CLS: KEY OFF: DEFINT C: DEFSTR H,S,M: H="0123456789ABCDEF": LOCATE ,,0
3 COLOR 4,0: PRINT: INPUT "MEMORY SEG (HEX or Null for BASIC Seg) >",S: A=0
4 IF S="" THEN DEF SEG: S="Basic": GOTO 6 ELSE S=">"+LEFT$("0000",4-LEN(S))+S
5 FOR I=2 TO 5: A=A+(INSTR(H,MID$(S,I,1))-1)*16^(5-I): NEXT: DEF SEG=A
6 PRINT "MEMORY SEG = "+S,A,A*16: PRINT: INPUT "START ADDRESS, # OF BYTES ",A,B
7 FOR A=A TO A+B STEP 10: M="": COLOR 4: PRINT USING "&: #####   ";S;A;: COLOR 6
8 FOR B=0 TO 9: C=PEEK(A+B): PRINT USING "### ";C;: IF C<32 THEN C=46
9 M=M+CHR$(C): NEXT: PRINT TAB(70);M: IF INKEY$="" THEN NEXT: GOTO 3 ELSE 3
```

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

Back issues are available. We can start your subscription with the FEB. 84 issue
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

THE SMART PROGRAMMER