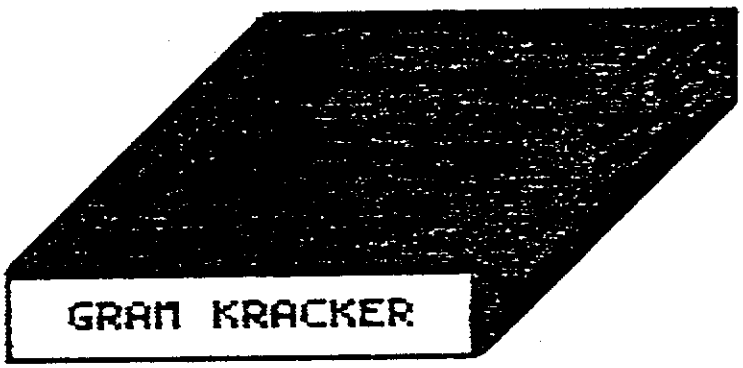
KRACKER FACTS



THE LITTLE BOX THAT COULD

COMPILED FOR THE LA 99ERS

BY MIKE DODD

LA 99ERS COMPUTER GROUP

P.O.B. 3547 GARDENA CA 90247

TABLE OF CONTENTS

INTRODUCTION TO KRACKER FACTS

Ever since the GRAM Kracker was released in late 1985, people have come up with many changes to the operating system and the cartridges. Some were uploaded to CompuServe, some to GEnie, some were published in newsletters, and some were just passed around by word of mouth. Unfortunately, there was no one place that someone could look for all of the changes.

When MG released Danny Michael's excellent GK Utility I disk, it was very helpful - many changes on one disk, ready to run. But there were still many changes that people had made, and they were scattered all over the four corners of the TI community.

This booklet, Kracker Facts, is an attempt by the Los Angeles 99er Users' Group to assemble all of the articles and modifications for the GRAM Kracker in one publication.

In here are articles by Tom Freeman, Millers Graphics (Craig Miller and D.C. Warren), Mike Dodd, and Walt Howe. All are targeted towards getting more out of your GRAM Kracker. We hope you enjoy them.

A LITTLE INTRODUCTION TO GPL CODE
by Craig Miller (MG)

We thought you might like to see what a powerful and compact language GPL code is. With the GRAM KRACKER and a GPL Assembler you will be able to write programs that can reside in the Module space and will be displayed on your Main Menu as a selection. GPL can also link to Assembly and BASIC programs! So you will have FULL use of the THREE built-in languages in our 4As (Basic, GPL and Assembly). Eat your hearts out all you Atari, Commodore, IBM and other computer owners!

```
*---------------------------------------------------------------------*
          * Disassembly of part of the Editor/Assembler Module *
                  * Starting at Grom >6069 thru >6132 *
*---------------------------------------------------------------------*

>6069  MOVE  7 FROM G@REGDAT TO VR01       Load the Vdp registers

       CALL  CHKMEM                        Go check for memory expansion and
                                           load the (C) character data

       MOVE  16 FROM G@CURSOR TO V@>08F0   Load the box and solid cursor data

* Put up the first Menu Screen

       ST    >7E,@SUBSTK                  Initialize the Sub Return stack pointer
       DCLR  @ERRCODE                     Zero out A/L Error Code Indicator
       DCLR  @GROMFLG                     Zero the Grom Flag
       ALL   SPACE                        Clear the screen with space characters

       FMT                                Start formatted screen output
       ROW   2                            At row 2
```

```
        COL   1                          At column 1 (note 0,0 is home position)
        HTEXT '* EDITOR/ASSEMBLER * '    Put up horizontal text
        ROW+  2                          At current row plus 2
        COL   1                          At column 1
        HTEXT 'PRESS:'                   .
        ROW+  2                          .
        COL   2                          .
        HTEXT '1 TO EDIT'                .  etc.
        ROW+  2                          Note: VTEXT, HCHAR, VCHAR are also
        COL   2                          allowed in a FMT, so is
        HTEXT '2    ASSEMBLE'            FOR xx - where xx equals
        ROW+  2                          the repeat loop counter
        COL   2
        HTEXT '3    LOAD AND RUN'
        ROW+  2
        COL   2
        HTEXT '4    RUN'
        ROW+  2
        COL   2
        HTEXT '5    RUN PROGRAM FILE'
        ROW+  6
        COL   2
        HTEXT >0A                        >0A is the (C) character
        HTEXT '1981 TEXAS INSTRUMENTS'
        FEND                             End the formatted screen output

GETKY   SCAN                             Scan the keyboard for a key press
        BR    GETKY                      BR (Branch on Reset) no NEW key pressed
        CEQ   FCTN9,@KEY                 Was FCTN 9 (Back) Pressed
        BR    GETKY1                     NO! check the other keys
        EXIT                             YES! Execute the Power Up routine
GETKY1  SUB   >31,@KEY                   Subtract >31 from the keycode (0 - ?)
        CHE   >05,@KEY                   If it's now Higher than 4 - wrong key
        BS    GETKY                      So, go wait for another key press

        CASE  @KEY                       Otherwise if @KEY equals

        BR    EDIT                       0 - goto Edit Menu

        BR    ASSEM                      1 - goto Load Assembler Prompt

        BR    LODRUN                     2 - goto Load and Run prompt

        BR    RUN                        3 - goto Run Program prompt

        BR    RUNPRG                     4 - goto Run Program File prompt
```

Notes:

The above code only requires 202 bytes of memory and that includes 119 bytes of text! So that means the actual instruction code only uses 83 bytes of memory! There isn't another language available for our 99/4As that is as compact as GPL. And, when compared to Assembly, it is much easier to program in. This is THE Language that TI should have released to us in the first place!

Most instructions can work with bytes or words. The D in front of an instruction indicates a word operation. The first operand to is SOURCE and the second is the DESTINATION. ie: ST >03,@TEMP1 stores one byte with the value of 3 into location TEMP1.

The COND bit in the GPL Status register (>837C) is turned ON if the test is TRUE and OFF when FALSE. It is also turned on when a NEW key is pressed on a keyboard scan

or when the result of certain instructions is zero.

BR = Branch On Reset... or Branch if the COND bit in the GPL Status register is OFF

BS = Branch On Set..... or Branch if the COND bit in the GPL Status register is ON

CASE is like ON X GOTO ..... except it starts at zero instead of 1 (Note: the COND bit is always turned OFF (reset) for a CASE or DCASE)

A CALL works like a GOSUB or Assembly's BL (Branch and Link)

'ALL' fills the screen with the one byte character following the instruction. (That's right only 2 bytes to clear the screen!!!!)

MOVE is a very powerful GPL instruction. With it you can MOVE x number of bytes FROM any type of memory TO any type of memory You can also move bytes to the VDP Registers! The MOVE instruction only requires 6 to 7 bytes for its object code!

SCAN (to scan the keyboard) only requires 1 byte of object code!!! (SCAN = >03)

Speed Test:

We ran the old 1 to 10,000 timing test in GPL to see how it compares to the other languages and here is how it came out.

1. In an incrementing loop with a DCEQ (double Compare Equal) 6.8 seconds.
2. In a decrementing loop [no compare just BR (not zero)] 4.3 seconds.

As we have seen from previous tests this places third on the list.

1. Assembly - well under .5 second
2. Forth - approx 1.3 seconds
3. GPL - 4.3 to 6.8 seconds
4. Pascal - I think this is where it falls
4. XB - 33.9 seconds
5. Basic - weeks (just kidding)

Since its not as fast as Assembly or Forth you are probably wondering why we are so excited about GPL?! True, a CRAY 3 it's not. However, it requires LESS THAN one half the space of Assembly code! With the Gram Kracker you have up to 58K of GPL program space (with 6K reserved for the Operating System), which would require AT LEAST 116K of Assembly code. This still leaves ALL of memory expansion free plus the 16K of cartridge RAM free for other things or for Assembly routines for your GPL programs to link to (another 48K). That gives us a TOTAL program space of 106K plus 16K of VDP Ram for a total of 122K (128K with the Operating System area). Also with GPL you can EXPAND or modify existing Modules. And, last but certainly not least, GPL is the controlling language for our 4As, so now you make it do most anything you want! Start thinking about those changes you've wanted to make for the last 6 years, your chance is coming!!!

AN EXPLANATION OF THE GPL XML INSTRUCTION
by Craig Miller (MG)

If you are using Gram to store an Assembly file in that is MOVEd out by a CALL or a GPL program (patch) you can start the Assembly program with a GPL XML statement.

The Opcode for GPL XML is >0F xx - where xx represents the XML table to use for the start vector (See the Explorer Manual page 77 for the XML tables). For example let's say you used a GPL MOVE to move an 8K assembly program out of Gram 7 (>E000) to high Memory Expansion and now you want to go out of GPL and execute your Assembly program. Let's say that your Assembly program starts at address >A040, this could be

the code you could use to do this task.

```
31 20 00          MYPROG    MOVE >2000,G@>E000,@>A000
8F 1D 00
E0 00

BF 00 A0                    DST  >A040,@>8300       (store start address)
40

0F F0                       XML >F0    (go to >8300 to get start address)

00                          RTN
```

When your Assembly program is finished you can then B @>006A to go back to the GPL Interpreter. Don't forget to reset the Grom Address if your Assembly program changed it. When the GPL Interpreter starts back up it will grab the >00 opcode (RTN) and return from the CALL MYPROG that you set up somewhere else in Gram to start the above routine. By the way, the Opcode for a CALL is >06 so the CALL MYPROG would be 06 xx xx where xx xx = the address in Gram where you placed the above code.

PROGRAMMING EXAMPLES OF THE GPL "MOVE" INSTRUCTION
by Craig Miller (MG)

Listed below are a number of examples of the GPL MOVE statement. This is a LIST file generated by the GPL Assembler.

When the GPL Interpreter talks to CPU Memory it offsets the CPU address by >8300. This can be seen in the OPCODES for the third move statement which breaks down as follows:

```
>35         MOVE
>1234       >1234 bytes
>8F         to CPU Memory (non-indexed) (>AF = VDP memory)
>9D00       at >2000 (>9D00+>8300=>2000)
>8F         from CPU Memory (non-indexed)
>1D00       at >A000 (>1D00+>8300=>A000)
```

When the GPL Interpreter talks to CPU Scratch Pad Memory Below >8380 or when a Scratch Pad address is used for indexing it is referenced by one byte (i.e. >831F will appear as >1F in the Opcode).

```
99/4 GPL-ASSEMBLER (Pass 1) correct                          PAGE 0001
GROM 3 - MOVE TEST
<0001>
<0002>                      GROM 3
<0003>                      AORG 0
<0004>
<0005>              * GPL   MOVE STATEMENT
<0006>              *
<0007>              *       MOVE #bytes,source,destination
<0008>              *
<0009> 6000 21,12,34        MOVE >1234,G@>C000,G@>E000
       6003 E0,00,C0
```

```
       6006 00
<0010> 6007 35,12,34          MOVE >1234,V@>1000,V@>3000
       600A AF,30,00
       600D AF,10,00
<0011> 6010 35,12,34          MOVE >1234,@>A000,@>2000
       6013 8F,9D,00
       6016 8F,1D,00
<0012> 6019 35,12,34          MOVE >1234,@>A000,@>831F
       601C 1F,8F,1D
       601F 00
<0013> 6020 35,12,34          MOVE >1234,@>A000,@>839E
       6023 80,9E,8F
       6026 1D,00
<0014>
<0015> 6028 31,12,34          MOVE >1234,G@>C000,V@>3000
       602B AF,30,00
       602E C0,00
<0016> 6030 31,12,34          MOVE >1234,G@>C000,@>2000
       6033 8F,9D,00
       6036 C0,00
<0017> 6038 31,12,34          MOVE >1234,G@>C000,@>831F
       603B 1F,C0,00
<0018> 603E 31,12,34          MOVE >1234,G@>C000,@>839E
       6041 80,9E,C0
       6044 00
<0019>
<0020> 6045 25,12,34          MOVE >1234,V@>1000,G@>C000
       6048 C0,00,AF
       604B 10,00
<0021> 604D 35,12,34          MOVE >1234,V@>1000,@>2000
       6050 8F,9D,00
       6053 AF,10,00
<0022> 6056 35,12,34          MOVE >1234,V@>1000,@>831F
       6059 1F,AF,10
       605C 00
<0023> 605D 35,12,34          MOVE >1234,V@>1000,@>839E
       6060 80,9E,AF
       6063 10,00
<0024>
<0025> 6065 25,12,34          MOVE >1234,@>2000,G@>C000
       6068 C0,00,8F
       606B 9D,00
<0026> 606D 35,12,34          MOVE >1234,@>2000,V@>1000
       6070 AF,10,00
       6073 8F,9D,00
<0027> 6076 35,12,34          MOVE >1234,@>831F,@>832F
       6079 2F,1F
<0028> 607B 35,12,34          MOVE >1234,@>839E,@>83AE
       607E 80,AE,80
       6081 9E
<0029> 6082 35,12,34          MOVE >1234,@>839E,@>831F
       6085 1F,80,9E
<0030>
<0031>                  * INDEXED MOVES -------------------------------
<0032>
<0033> 831F             TEMP1  EQU  >831F
<0034> 839E             TEMP2  EQU  >839E
<0035>
<0036> 6088 29,12,34          MOVE >1234,G@>C000,G@2(@TEMP2)
```

```
       608B 00,02,9E
       608E C0,00
<0037> 6090 31,12,34          MOVE >1234,G@>C000,V@2(@TEMP2)
       6093 E0,02,9E
       6096 C0,00
<0038> 6098 31,12,34          MOVE >1234,G@>C000,@2(@TEMP2)
       609B CF,7D,02
       609E 9E,C0,00
<0039> 60A1 31,12,34          MOVE >1234,G@>C000,@>830F(@TEMP1)
       60A4 C0,0F,1F
       60A7 C0,00
<0040> 60A9 2B,12,34          MOVE >1234,G@1(@TEMP1),G@2(@TEMP2)
       60AC 00,02,9E
       60AF 00,01,1F
<0041>
<0042> 60B2 2D,12,34          MOVE >1234,@>A000,G@2(@TEMP2)
       60B5 00,02,9E
       60B8 8F,1D,00
<0043> 60BB 35,12,34          MOVE >1234,V@1(@TEMP1),V@2(@TEMP2)
       60BE E0,02,9E
       60C1 E0,01,1F
<0044> 60C4 35,12,34          MOVE >1234,@1(@TEMP1),@2(@TEMP2)
       60C7 CF,7D,02
       60CA 9E,CF,7D
       60CD 01,1F
<0045>
<0046>                  * INDIRECT MOVES ----------------------------
<0047>
<0048> 60CF 31,12,34          MOVE >1234,G@>C000,V*TEMP2
       60D2 B0,9E,C0
       60D5 00
<0049> 60D6 31,12,34          MOVE >1234,G@>C000,*TEMP2
       60D9 90,9E,C0
       60DC 00
<0050> 60DD 31,12,34          MOVE >1234,G@>C000,*>830F(@TEMP1)
       60E0 D0,0F,1F
       60E3 C0,00
<0051> 60E5 33,12,34          MOVE >1234,G@1(@TEMP1),*TEMP2
       60E8 90,9E,00
       60EB 01,1F
<0052>
<0053> 60ED 35,12,34          MOVE >1234,V*1(@TEMP1),V*2(@TEMP2)
       60F0 FF,7D,02
       60F3 9E,FF,7D
       60F6 01,1F
<0054> 60F8 35,12,34          MOVE >1234,*TEMP1,*TEMP2
       60FB 90,9E,90
       60FE 1F
<0055>
<0056>                        END
```

"CALL CAT" GPL SOURCE CODE
by Craig Miller (MG)

The following file is a LIST file from the GPL Assembler. We uploaded it to give you an example of a GPL program that is on the final Gram Kracker Utility diskette. This is a new CALL for Extended Basic that will patch itself to XB version 110. The call is CALL CAT("DSK1.") to catalog the floppy in drive 1. This cataloger will also support other divices that contain a "CATALOG" routine such as the MYARC Hard Disk and the MYARC RAM disk.

By comparing the OPCODEs in the third column with other Grom/Gram code you should be able find out what is going on in other modules and in Grom 0.

Hope this file helps you understand GPL a little more. Have fun.

```
TI99/4 GPL-ASSEMBLER

GROM 6 - XB Cat 12,17,85

<0001>                      GROM 6
<0002>                      AORG >1C00        * Routine loads at GRAM >DC00
<0003>
<0004>              *
<0005>              * Absolute equates into version 110 X-BASIC cartridge
<0006>              *
<0007> 6A78         CHKEND EQU  >6A78         Routine to check end of statement
<0008> 6D78         ERR    EQU  >6D78         Error routine
<0009> C533         ERRSYN EQU  >C533         SYNTAX error
<0010> C592         ERRCIP EQU  >C592         COMMAND ILLEGAL IN PROGRAM error
<0011> C59A         ERRBA  EQU  >C59A         BAD ARGUMENT error
<0012>              *
<0013>              * PAD equates
<0014>              *
<0015> 8304         PABPTR EQU  >8304         PAB pointer register
<0016> 8310         TEMP   EQU  >8310         Temporary registers
<0017> 8312         TEMP1  EQU  >8312            .
<0018> 8314         TEMP2  EQU  >8314            .
<0019> 8342         CHAT   EQU  >8342         Last character register
<0020> 8344         RUN    EQU  >8344         Running program flag
<0021> 8356         NMPNTR EQU  >8356         DSR name length pointer
<0022> 8375         KEY    EQU  >8375         Key code returned by key scan
<0023>              *
<0024>              * XML equates into X-BASIC cartridge
<0025>              *
<0026> 0073         CNS    EQU  >73           Convert floating to string
<0027> 0074         PARSE  EQU  >74           Parse routine
<0028> 0079         PGMCH  EQU  >79           Advance character routine
<0029> 0083         SCROLL EQU  >83           Screen scroll routine
<0030>              *
<0031>              * VDP equates
<0032>              *
<0033> 0820         PAB    EQU  >0820         PAB. Crunch buffer area
<0034> 0836         VBUFF  EQU  >0836         Buffer location
<0035> 0828         VLENA  EQU  >0828         File name length in crunch buffer
<0036> 0829         VLENB  EQU  >0829         File name length in PAB
<0037> 08CA         RCLBUF EQU  >08CA         Recall buffer address
<0038>              *
<0039>              * Misc. equates
<0040>              *
```

```
<0041> 00B6          RPAR   EQU  >B6          Right paren. token
<0042> 00B7          LPAR   EQU  >B7          Left paren. token
<0043> 020D          READ   EQU  >020D        DSR read code
<0044> 010D          CLOSE  EQU  >010D        DSR close code
<0045> 0020          SPACE  EQU  >20          Space char.
<0046> 0002          FCTN4  EQU  >02          CLEAR char.
<0047> 0012          RRTN   EQU  >0012        GROM 0 return routine
<0048>               *
<0049>               ***************************************************************
<0050>               * X-BASIC DEVICE CATALOGER
<0051>               *  Loads at GRAM address >DC00
<0052>               *  Accessed with a CALL
<0053>               *  PAB is installed in Crunch buffer area
<0054>               *
<0055>               *
<0056>               *  D.C. Warren 12/17/85
<0057>               ***************************************************************
<0058>               *
<0059> DC00 8E,44    CAT    CZ   @RUN             Is a program running?
<0060> DC02 45,92           BR   ERRCIP           YES! Error so tell user
<0061>
<0062> DC04 D6,42,B7        CEQ  LPAR,@CHAT       Do we have a '(' ?
<0063> DC07 45,33           BR   ERRSYN           NO! SYNTAX error
<0064>
<0065> DC09 0F,79           XML  PGMCH            Advance program pointer
<0066>
<0067> DC0B 0F,74           XML  PARSE            Parse to ')'
<0068> DC0D B6              BYTE RPAR *
<0069>
<0070> DC0E D6,4C,65        CEQ  >65,@FAC+2       Do we have a string?
<0071> DC11 45,9A           BR   ERRBA            NO! Bad Argument
<0072>
<0073> DC13 8F,50           DCZ  @FAC+6           Is it a null string?
<0074> DC15 65,9A           BS   ERRBA            YES! Bad Argument
<0075>
<0076> DC17 C6,51,0B        CH   11,@FAC+7        Don't allow device name
<0077> DC1A 65,9A           BS   ERRBA             greater than 11 chars.
<0078>
<0079> DC1C D6,42,B6        CEQ  RPAR,@CHAT       Last char a ')'?
<0080> DC1F 45,33           BR   ERRSYN           NO! Syntax error
<0081>              *
<0082>              * Set up PAB at V>820
<0083>              *  The next 7 lines move the name over one byte!!
<0084>              *
<0085> DC21 86,10           CLR  @TEMP
<0086> DC23 BC,11,A8        ST   V@VLENA,@TEMP+1  Get name length
       DC26 28
<0087> DC27 BD,14,10        DST  @TEMP,@TEMP2     Save name length
<0088> DC2A 91,10           DINC @TEMP            Adjust TEMP
<0089>
<0090> DC2C 35,00,01 CAT1   MOVE 1,V@VLENA-1(@TEMP),V@VLENA(@TEMP) Move a
       DC2F E8,28,10
       DC32 E8,27,10
<0091>                                             * byte over
<0092> DC35 93,10           DDEC @TEMP            Keep going until whole
<0093> DC37 5C,2C           BR   CAT1              name is moved
<0094>
<0095> DC39 31,00,09        MOVE 9,G@PABDAT,V@PAB Install PAB
       DC3C A8,20,DE
```

```
      DC3F 38
<0096>                *
<0097>                * Open Device
<0098>                *
<0099> DC40 06,DE,23        CALL DSRER              Link to device
<0100>                *
<0101>                * Read first record
<0102>                *
<0103> DC43 BF,A8,20        DST  READ,V@PAB         Make PAB a read
       DC46 02,0D
<0104> DC48 06,DE,23 CAT2   CALL DSRER              Link to device
<0105>                *
<0106>                * Put disk information on the screen
<0107>                *
<0108> DC4B 07,80           ALL  >80                Clear screen
<0109> DC4D 34,14,A2        MOVE @TEMP2,V@RCLBUF,V@>282 Put device name up
       DC50 82,A8,CA
<0110>
<0111> DC53 08              FMT
<0112> DC54 FC,60            SCRO >60
<0113> DC56 FE,14            ROW  20
<0114> DC58 FF,09            COL  09
<0115> DC5A 09,20,44         HTEX  ' Diskname='
       DC5D 69,73,6B
       DC60 6E,61,6D
       DC63 65,3D
<0116>
<0117> DC65 A0               ROW+  1
<0118> DC66 FF,02            COL   2
<0119> DC68 15,41,76         HTEX  'Available=       Used='
       DC6B 61,69,6C
       DC6E 61,62,6C
       DC71 65,3D,20
       DC74 20,20,20
       DC77 20,20,20
       DC7A 55,73,65
       DC7D 64,3D
<0120>
<0121> DC7F A0               ROW+  1
<0122> DC80 FF,02            COL   2
<0123> DC82 1C,20,46         HTEX ' Filename  Size    Type     P'
       DC85 69,6C,65
       DC88 6E,61,6D
       DC8B 65,20,20
       DC8E 53,69,7A
       DC91 65,20,20
       DC94 20,20,54
       DC97 79,70,65
       DC9A 20,20,20
       DC9D 20,20,50
<0124>
<0125> DCA0 A0               ROW+  1
<0126> DCA1 FF,02            COL   2
<0127> DCA3 1C,2D,2D         HTEX '---------- ---- ----------- -'
       DCA6 2D,2D,2D
       DCA9 2D,2D,2D
       DCAC 2D,2D,20
       DCAF 2D,2D,2D
       DCB2 2D,20,2D
```

```
       DCB5 2D,2D,2D
       DCB8 2D,2D,2D
       DCBB 2D,2D,2D
       DCBE 2D,20,2D
<0128> DCC1 FB              FEND
<0129>
<0130>                *
<0131>                * Put disk name on screen
<0132>                *
<0133> DCC2 06,DE,00        CALL DISSTR          Get string into FAC
<0134> DCC5 8E,4B           CZ   @FAC+1          Skip if zero length
<0135> DCC7 7C,D3           BS   CAT3
<0136>
<0137> DCC9 08              FMT
<0138> DCCA FC,60            SCRO >60            Put disk name on screen
<0139> DCCC FE,14            ROW   20              .
<0140> DCCE FF,14            COL   20              .
<0141> DCD0 E9,4C            HSTR 10,@FAC+2        .
<0142> DCD2 FB              FEND                   .
<0143>                *
<0144>                * Display AVAILABLE device space on screen
<0145>                *
<0146> DCD3 A1,10,4A CAT3   DADD @FAC,@TEMP      Go to next field
<0147> DCD6 A3,10,00        DADD 19,@TEMP        Continue to last field
       DCD9 13
<0148> DCDA BF,14,02        DST  >2AC,@TEMP2     Set up screen address
       DCDD AC
<0149> DCDE 06,DD,E4        CALL DISNUM          Display AVAILABLE space
<0150>                *
<0151>                * Display USED device space on the screen
<0152>                *
<0153> DCE1 A7,10,00        DSUB 9,@TEMP         Point to FORMATTED space
       DCE4 09
<0154> DCE5 35,00,08        MOVE 8,V*TEMP,@ARG   Move it into ARG
       DCE8 5C,B0,10
<0155> DCEB 0F,07           XML  FSUB            Developed USED value
<0156> DCED BF,14,02        DST  >2B8,@TEMP2     Set up screen address
       DCF0 B8
<0157> DCF1 06,DD,EA        CALL DISNU1          Display USED space
<0158>
<0159>                *
<0160>                * List catalog
<0161>                *
<0162> DCF4 03       CAT4   SCAN                 Scan the keyboard
<0163> DCF5 5D,0E           BR   CAT4B           Continue if no new key
<0164> DCF7 D6,75,02        CEQ  FCTN4,@KEY      CLEAR key?
<0165> DCFA 7D,C2           BS   DONE            YES! Abort
<0166> DCFC D6,75,20        CEQ  SPACE,@KEY      SPACE key?
<0167> DCFF 5D,0E           BR   CAT4B           NO! Keep going
<0168>
<0169> DD01 03       CAT4A  SCAN                 Scan keyboard
<0170> DD02 5D,01           BR   CAT4A           Loop until new key press
<0171> DD04 D6,75,02        CEQ  FCTN4,@KEY      CLEAR?
<0172> DD07 7D,C2           BS   DONE            YES! Abort
<0173> DD09 D6,75,20        CEQ  SPACE,@KEY      SPACE key?
<0174> DD0C 5D,01           BR   CAT4A           NO! Continue to wait
<0175>
<0176> DD0E 0F,83    CAT4B  XML  SCROLL           Scroll the screen
<0177>
```

```
<0178> DD10 06,DE,23        CALL DSRER          Link to device
<0179>
<0180> DD13 06,DE,00        CALL DISSTR         Get string into FAC
<0181> DD16 8E,4B           CZ   @FAC+1         Skip display if zero
<0182> DD18 7D,24           BS   CAT5            length
<0183>
<0184> DD1A 08              FMT
<0185> DD1B FC,60            SCRO >60           Put disk name on screen
<0186> DD1D FE,17            ROW   23                .
<0187> DD1F FF,02            COL   02                .
<0188> DD21 E9,4C            HSTR 10,@FAC+2          .
<0189> DD23 FB              FEND                     .
<0190>
<0191> DD24 A1,10,4A CAT5   DADD @FAC,@TEMP     Go to next field
<0192> DD27 A3,10,00        DADD 10,@TEMP       Continue another field
       DD2A 0A
<0193> DD2B 8F,B0,10        DCZ  V*TEMP         Time to get out if
<0194> DD2E 7D,C2           BS   DONE            zero file size
<0195> DD30 BF,14,02        DST  >2EC,@TEMP2    Set up screen address
       DD33 EC
<0196> DD34 06,DD,E4        CALL DISNUM         Display file length
<0197>
<0198> DD37 A7,10,00        DSUB 9,@TEMP        Back a field
       DD3A 09
<0199> DD3B 35,00,08        MOVE 8,V*TEMP,@FAC  Move it into FAC
       DD3E 4A,B0,10
<0200> DD41 0F,12           XML  CFI            Convert it to an int.
<0201>
<0202> DD43 8E,4A           CZ   @FAC           Non-negative?
<0203> DD45 7D,4D           BS   CAT5A          YES! File not protected
<0204>
<0205> DD47 BE,A2,FE        ST   >B9,V@>2FE     Put a 'Y' on screen
       DD4A B9
<0206> DD4B 83,4A           DNEG @FAC           Make number positive
<0207>
<0208> DD4D 92,4B    CAT5A  DEC  @FAC+1         Adjust for CASE
<0209>
<0210> DD4F 8A,4B           CASE @FAC+1         Show file type
<0211> DD51 5D,5B           BR   DF                 .
<0212> DD53 5D,6D           BR   DV                 .
<0213> DD55 5D,7F           BR   IF                 .
<0214> DD57 5D,91           BR   IV                 .
<0215> DD59 5D,A3           BR   PR                 .
<0216>
<0217> DD5B 08        DF    FMT
<0218> DD5C FC,60            SCRO >60
<0219> DD5E FE,17            ROW   23
<0220> DD60 FF,12            COL   18
<0221> DD62 06,44,69         HTEX 'Dis/Fix'
       DD65 73,2F,46
       DD68 69,78
<0222> DD6A FB              FEND
<0223> DD6B 5D,B5           BR   CAT6
<0224>
<0225> DD6D 08        DV    FMT
<0226> DD6E FC,60            SCRO >60
<0227> DD70 FE,17            ROW   23
<0228> DD72 F,12            COL   18
<0229> DD74 06,44,69         HTEX 'Dis/Var'
```

```
       DD77 73,2F,56
       DD7A 61,72
<0230> DD7C FB              FEND
<0231> DD7D 5D,B5           BR   CAT6
<0232>
<0233> DD7F 08        IF    FMT
<0234> DD80 FC,60            SCRO >60
<0235> DD82 FE,17            ROW   23
<0236> DD84 FF,12            COL   18
<0237> DD86 06,49,6E         HTEX 'Int/Fix'
       DD89 74,2F,46
       DD8C 69,78
<0238> DD8E FB              FEND
<0239> DD8F 5D,B5           BR   CAT6
<0240>
<0241> DD91 08        IV    FMT
<0242> DD92 FC,60            SCRO >60
<0243> DD94 FE,17            ROW   23
<0244> DD96 FF,12            COL   18
<0245> DD98 06,49,6E         HTEX 'Int/Var'
       DD9B 74,2F,56
       DD9E 61,72
<0246> DDA0 FB              FEND
<0247> DDA1 5D,B5           BR   CAT6
<0248>
<0249> DDA3 08        PR    FMT
<0250> DDA4 FC,60            SCRO >60
<0251> DDA6 FE,17            ROW   23
<0253> DDAA 06,50,72         HTEX 'Program'
       DDAD 6F,67,72
       DDB0 61,6D
<0254> DDB2 FB              FEND
<0255> DDB3 5C,F4           BR   CAT4
<0256>
<0257> DDB5 A3,10,00 CAT6   DADD 18,@TEMP          Advavce two fields
       DDB8 12
<0258> DDB9 BF,14,02        DST  >2F9,@TEMP2       Set up screen address
       DDBC F9
<0259> DDBD 06,DD,E4        CALL DISNUM            Display record length
<0260> DDC0 5C,F4           BR   CAT4              Do it all again
<0261>
<0262> DDC2 0F,83    DONE   XML  SCROLL            One last scroll
<0263> DDC4 06,DE,1A        CALL CLSFL             Close file
<0264> DDC7 0F,79           XML  PGMCH             Parse past ')'
<0265> DDC9 06,6A,78        CALL CHKEND            SYNTAX error if not end
<0266> DDCC 45,33           BR   ERRSYN               .
<0267> DDCE 06,00,12        CALL RRTN              Return to X-BASIC
<0268>                *
<0269>                * File error
<0270>                *
<0271> DDD1           ERROR EQU  $
<0272> DDD1 BF,04,08        DST  PAB-4,@PABPTR     Fake a BASIC PAB
       DDD4 1C
<0273> DDD5 BD,10,A8        DST  V@PAB,@TEMP       Save error
       DDD8 20
<0274> DDD9 06,DE,1A        CALL CLSFL             Close file
<0275> DDDC BD,A8,20        DST  @TEMP,V@PAB       Restore error
       DDDF 10
<0276> DDE0 06,6D,78        CALL ERR               Return through ERR
```

```
<0277> DDE3 24              BYTE 36 *              I/O ERROR XX
<0278>                *
<0279>                ************************************************************
<0280>                * Subroutines
<0281>                ************************************************************
<0282>                *
<0283>                * Display number subroutine
<0284>                *  ENTER: Floating number in FAC for DISNU1
<0285>                *         Screen address in TEMP2
<0286>                *
<0287> DDE4 35,00,08 DISNUM MOVE 8,V*TEMP,@FAC     Move FLP number to FAC
       DDE7 4A,B0,10
<0288> DDEA 86,55    DISNU1 CLR  @FAC+11           Indicate a free format
<0289> DDEC 0F,73           XML  CNS               Convert FAC to a string
<0290> DDEE A2,90,55 DISNU2 ADD  >60,*FAC+11       Add offset to string
       DDF1 60
<0291> DDF2 BC,B0,14        ST   *FAC+11,V*TEMP2   Put a char on the screen
       DDF5 90,55
<0292> DDF7 91,14           DINC @TEMP2            Increment screen addr.
<0293> DDF9 90,55           INC  @FAC+11           Increment FAC addr.
<0294> DDFB 92,56           DEC  @FAC+12           Decrement string length count
<0295> DDFD 5D,EE           BR   DISNU2            Loop until done
<0296> DDFF 00              RTN                    Return to caller
<0297>                *
<0298>                * Prepare a VDP string for FORMAT statement
<0299>                *  LEAVE: FAC has string length (word)
<0300>                *         FAC+2 has string
<0301>                *         TEMP pointing to next string in record
<0302>                *
<0303> DE00 BF,10,08 DISSTR DST  VBUFF,@TEMP       Get buffer address
       DE03 36
<0304> DE04 86,4A           CLR  @FAC              Clear MSB of FAC word
<0305> DE06 BC,4B,B0        ST   V*TEMP,@FAC+1     Store disk name length
       DE09 10
<0306> DE0A 91,10           DINC @TEMP             Point to string
<0307>
<0308> DE0C BE,4C,20        ST   >20,@FAC+2        Clear out string space
<0309> DE0F 35,00,09        MOVE 9,@FAC+2,@FAC+3
       DE12 4D,4C
<0310> DE14 34,4A,4C        MOVE @FAC,V*TEMP,@FAC+2 Move disk name into FAC
       DE17 B0,10
<0311> DE19 00              RTN
<0312>                *
<0313>                * Close file
<0314>                *
<0315> DE1A BF,A8,20 CLSFL  DST  CLOSE,V@PAB       A close operation
       DE1D 01,0D
<0316> DE1F 06,DE,2F        CALL DSR               Link to device
<0317> DE22 00              RTN                    Return to caller
<0318>                *
<0319>                * DSR LINK with error handling
<0320>                *
<0321> DE23 06,DE,2F DSRER  CALL DSR
<0322> DE26 7D,D1           BS   ERROR             Branch on no-device
<0323> DE28 D6,A8,21        CEQ  >0D,V@PAB+1       Check for device errors
       DE2B 0D
<0324> DE2C 5D,D1           BR   ERROR
<0325> DE2E 00              RTN                    Return to caller
<0326>                *
```

```
<0327>                  * DSR LINK routine
<0328>                  *
<0329> DE2F BF,56,08 DSR    DST  VLENB,@NMPNTR     Name length pointer
       DE32 29
<0330> DE33 06,00,10        CALL >10               Call DSR
<0331> DE36 08              BYTE 8 *               DSR call
<0332> DE37 01              RTNC                   Return with COND bit
<0333>                  *
<0334>                  * PAB data
<0335>                  *
<0336> DE38 00,0D,08 PABDAT BYTE >00,>0D,>08,>36,>00,>00,>00,>00,>00
       DE3B 36,00,00
       DE3E 00,00,00
```

Symbol Table

DC00 CAT	DC2C CAT1	DC48 CAT2	DCD3 CAT3	DCF4 CAT4
DD01 CAT4A	DD0E CAT4B	DD24 CAT5	DD4D CAT5A	DDB5 CAT6
8342 CHAT	6A78 CHKEND	010D CLOSE	DE1A CLSFL	0073 CNS
DD5B DF	DDEA DISNU1	DDEE DISNU2	DDE4 DISNUM	DE00 DISSTR
DDC2 DONE	DE2F DSR	DE23 DSRER	DD6D DV	6D78 ERR
C59A ERRBA	C592 ERRCIP	DDD1 ERROR	C533 ERRSYN	0002 FCTN4
DD7F IF	DD91 IV	8375 KEY	00B7 LPAR	8356 NMPNTR
0820 PAB	DE38 PABDAT	8304 PABPTR	0074 PARSE	0079 PGMCH
DDA3 PR	08CA RCLBUF	020D READ	00B6 RPAR	0012 RRTN
8344 RUN	0083 SCROLL	0020 SPACE	8310 TEMP	8312 TEMP1
8314 TEMP2	0836 VBUFF	0828 VLENA	0829 VLENB	

EXTENDED BASIC AUTO-BOOT ("DSK1.LOAD") BYPASS PATCH

First LOAD Extended Basic into the Gram Kracker.

From the Gram Kracker menu select 5 Memory Editor. Then press FCTN = for HEX, FCTN 1 for the Gram Memory Window and then press FCTN 5 for SEARCH.

Type in >6300 for the START address and >6400 for the FINISH address. Press FCTN 9 to put the cursor in the Search String Input area and type in 86 A3 71 and then press FCTN S (left arrow) to put the cursor on the last byte to search for. Next press ENTER to start the Search.

For most Extended Basic modules this Hex string will be found at >63CD. We'll call that "address A". Now press FCTN 5 to leave SEARCH and then press FCTN 9 to put the cursor in the Memory Window. Turn off the Write Protect (turn it to Bank 1). Now change the first two bytes (86 A3) to 58 00. This is a BRANCH ON RESET to >7800 instruction.

Press FCTN 9 and change the Memory Window to g7800. You will see garbage here (UNLESS YOU HAVE PREVIOUSLY PUT SOMETHING IN THIS SPACE!!). The GROMs are only 6K in length so the bytes in the last 2K are "garbage wrap around" read by the Gram Kracker Save routine. So, it's a good area for adding routines to your modules.

Press FCTN 9 to put the cursor in the Memory Window and at the g7800 memory location, put in the following code:

```
86 A3 71          CLR V@>371       Clear Auto Load needed flag
03                SCAN             Scan the Keyboard
D6 75 20          CEQ >20,@>8375   Is the Space Bar pressed
```

```
Now take your "address A" and add 6 to it !
>63CD + 6 = >63D3                          ]
3D3                BS "address A" plus 6 bytes  YES! (Branch on Set)
[Take your "address A", add 3 to it and replace the first digit with 4]
[>63CD + 3 = 63D0 ..... change it to 43D0                              ]
43D0               BR "address A" plus 3 bytes   NO!  (Branch on Reset)
```

For a module with a >63CD "address A" your memory window should now look like this:

```
g7800
================================
86 A3 71 03 D6 75 20 63 D3 43 D0 xx
xx xx xx xx xx xx xx xx xx xx xx xx
                    xx = don't care
```

Now restore the Write Protect, return to the Gram Kracker menu and resave your module.

Now when you select EXTENDED BASIC you can bypass the auto-load command by holding down the space bar!! (No more DSK1.LOAD search)

NOTE: if you are using the GK Utility I version of Extended Basic, you do not need to make this change, as it is included in the GK Utility patches.

NOTES ON THE ROM/RAM SPACE AT >6000 - 7FFF

by Craig Miller (MG)

Some of the modules that contain ROM write to their memory space, >6000 - >7FFF, to switch banks or as a form of protection. If the module loaded into the Gram Kracker is of this type you MUST have the Write Protect switch in the Write Protect position in order to use them. One example of this is TI Extended Basic. It writes to >6000 to enable bank 1 and >6002 to enable bank 2 of its ROM memory.

Some of the software currently available that loads into a Super Cart, >6000 - >7FFF expects RAM in this area and as such will only work properly if the Write Protect switch is NOT in the Write Protect position. One example of this is the modified Super Bug that loads at >6000. This program sets its workspace in the >6000 - >7FFF area of memory.

Since you have manual control over the Bank 1 - Bank 2 switch it is possible to have 2 different 8K Assembly programs in the cartridge RAM area, >6000 - >7FFF. For example you could have the above mentioned Super Bug in Bank 1 and say a Screen Dump program, that loads into this area, in Bank 2. Then with the flip of a switch you could have one or the other appear on the menu without having to re-load it.

Here is some information on the Bank Switching of the 8K ROM/RAM cartridge space.

With the WRITE PROTECT ON a piece of software can write to:

>6000, >6004, >6008 ... >7FF8, >7FFC etc. to select Bank 1

>6002, >6006, >600A ... >7FFA, >7FFE etc. to select Bank 2

This is how Extended Basic bank swaps the upper 4K (>7000 - >7FFF) to get 12K out of an 8K space. This is also how the Atari modules do bank swapping to get 16K out of an 8K space.

The software you write can also do this with a CLR @>6000 for Bank 1 and a CLR @>6002 for Bank 2 - BUT WRITE PROTECTION MUST BE ON or the banks won't swap, you'll just clear the word at that address. Bank swapping is disabled when Write Protection is turned off so we could load this space without it swapping banks.

To see bank swapping work, go into the Gram Kracker and load Extended Basic. Next select 5 Memory Editor from the Gram Kracker Menu. Type in c6FF0 for the Memory address and press FCTN = for Hex. Press FCTN 9 to put the cursor in the Memory Window, make sure Write Protection is ON and press and hold down the 1 key. As the cursor moves across the screen you will see the address space from >7000 to >7FFF swap banks. In reality the entire 8K block is switching banks but the first 4K (>6000 - >6FFF) is the same in both banks. This gives the appearance that the last 4K is bank switching and simulates the 12K of Rom in the Extended Basic's banks.

CHANGING THE BEEP AND HONK SOUNDS
by Mike Dodd

To change the sounds of the beep and honk, go into the GRAM Kracker memory editor. Press FCTN 1 for GRAM, FCTN = for hex, and FCTN 5 to search. Type 0000 for the start, 1000 for the end. Press FCTN 9 to enter the search window and type 05 92 0A 01 9F (don't type the spaces). When it finds it (mine was at >047E), press FCTN 5 to leave the search, FCTN 9 to enter the memory window, enable bank 1, and change the 05 to a new number (I used 10).

For the honk sound, follow the same procedure, except this time search for 20 90 0A 01 9F. Mine was at >0489. Change the 20 to a new number (I used 25).

The best way to hear the new sounds is to press CTRL = to get out of the memory editor, press 1 for load module, FCTN 3 and ENTER. That way you will hear both the beep and the honk.

When you've set them to your liking, save GRAM 0 to disk.

TITLE SCREEN REDESIGN
by Walt Howe

With the help of the GRAM KRACKER manual, "TI99/4A INTERN" by Heiner Martin, and my own poking around, I have put together this partial guide to modifying GROM 0, particularly the title screen and character sets. I can see that a lot more than this can be done as I begin to unravel the Graphic Programming Language code contained in GROM 0, but this guide will concentrate on the changes that can be made by changing nothing more than data tables and text strings.

TEXT MODIFICATIONS:

Most of the text on the title screen and the following menu screen appears in a single string beginning at (or near) memory address g048F. The string begins with the copyright symbol (hex 0A). For the sake of illustration here, I will use the "@" in its place. The complete string is "@1981 TEXAS INSTRUMENTSHOME COMPUTER". The copyright character will not appear in the GRAM KRACKER editor in ASCII mode. You have to switch to hex mode to see the 0A character. The copyright symbol itself is defined at g0998 - more about this later. If you do not want to keep the copyright symbol, you can overwrite it with whatever character you want or even redefine the symbol. The top text line on the screen uses the 8th through 24th characters of the string. The second line uses characters 25 through 37. The bottom line on the screen uses characters 1 through 24. Count spaces as characters, of course, and notice that there are two spaces after "1981". The top two lines are repeated on the following menu screen. The main things to realize are that any modifications to the string at g048F will appear in three different places, and that your replacement string cannot be longer than the given one. Other text appears as follows:

g014B - READY-PRESS ANY KEY TO BEGIN
g025D - PRESS
g094D - FOR

GRAPHICS CHANGES:

The Texas Instruments logo - the state of Texas with the embedded "t" and "i" - is defined beginning at or near g0950. Nine special graphics characters are designed which fit together in a 3x3

pattern to create the logo. The pattern is as follows:

123
456
789

The logo appears on the title screen, the menu screen, and is sometimes used by cartridge based programs, as well. If you substitute your own design, be prepared to find it appearing in unexpected places. The nine characters are defined by eight hex character pairs each or by 16 hex characters just as they are in basic/xbasic. In case you have one of the slightly different operating systems, look at or near g0950 for hex characters beginning 01 03 03 03 03 03 03 03 FC... .

Immediately after the logo patterns appear 8 hex pairs at or near g0998 defining the copyright sign. This pattern begins 3C 42 99 A1... text character in your own character string, or substitute your own pattern for your own purposes. It is identified in text by the hex pair 0A. It will not show up on screen in the GRAM KRACKER editor ASCII mode - only the hex mode.

EDITING COLORS & COLOR BARS:

The color table for the title screen and follow-on menu screen is located at or near g0459, beginning with a series of 12 hex 17's. The 17's define the character set colors (black on cyan). You can, of course, change these to any other preferred text and background colors. Following the 17's, the next 16 hex pairs, all beginning with 0, define the different colors that appear in the color bars. Change these to substitute your own color patterns as you wish. If you make them all the same color, the bars will be a solid color instead of a pattern of colored squares, for example. Whatever you select will appear in both the top and bottom color bars. Finally, the edge color is defined as the second digit of hex location g0458, which is F7. Change the 7 (cyan) to anything else you want.

CHARACTER SETS:

There are three character sets in GROM 0 - the large eight dot high capitals (with numbers and symbols - ASCII 32 through 95 or hex >20 through >6F), the 7-dot high capitals (likewise), and the so-called lower case characters, which are really small capitals. The NEWCHARS utility provided with the GRAM KRACKER alters the last two sets, but not the title screen capitals set. The eight dot set begins at g04B4 with a series of 8 00's, which is the space character, of course (ASCII 32 or hex >20). The smaller capitals begin immediately following the large capitals at g0684 with 7 00's for the space character. The lower case begins at g0874 with 00 20 10 08 00 00 00 representing the grave accent (`) or ASCII character 96 (>60) and continuing through character 127 (>7F). The set concludes at g094C, just before text "FOR" and the TI logo set.

SUMMARY OF KEY ADDRESSES:

HEX ADDR	BEGINS WITH	TEXT OR PURPOSE
014B	52 45 41 44	READY-PRESS ANY KEY
025D	50 52 45 53	PRESS
0458	F7	7 is cyan edge color
0459	17 17 17 17	Black on cyan chars.
0466	06 03 01 0B	Color bar colors
048F	0A 31 39 38	1981 TEXAS INSTRUM
04B4	00 00 00 00	Large capital set.
0684	00 00 00 00	Regular capital set.
0874	00 20 10 08	Lower case char set.
094D	46 4F 52	FOR
0950	01 03 03 03	TI logo definition
0998	3C 42 99 A1	Copyright definition

TO EXPLORE FURTHER:

It is fairly easy to move the color bars, change their size, and change and move text and graphics, but the systems of numbering screen locations are complex and far from obvious at first look (yes, I meant systems.) One of the systems is the consecutive numbering of locations in hex that is used in Assembly language. Another is to specify row and column addresses, but the addresses as they appear in hex code (the way you see it from the GRAM KRACKER) are a different story. Row addresses begin with A0 and column addresses begin with 80. A third system is to specify row and column offsets from the last address. If you have the book "TI99/4A INTERN", this should be enough to help you figure out the addressing systems. If you do not, I don't advise your trying to touch this area unless you are a very knowledgable programmer. To explain the uses of the different systems used by the GPL would approach book length (and I hope someone writes it!).

CHANGING THE KEYBOARD
by Mike Dodd

With the GRAM Kracker, you can finally change the keys on the 99/4A. One productive use of modifying the keyboard is to add printer codes - add keys for consensed, ESCAPE (ASCII 27), enlarged, etc. That way, while in console Basic or XB, you can type a PRINT #1:" command and type the keys, rather than having to use CHR$ statements.

Probably the best way to add new keys is to change the SHIFT, FCTN, and CTRL key codes for the SPACE and ENTER keys. TI left the ASCII codes the same for those two keys in all the modes. Here are the addresses, in GROM 0, of the SPACE and ENTER combinations.

KEY	SPACE	ENTER
FCTN	1766	1765
CTRL	1796	1795
SHIFT	1736	1735

If they aren't right at those locations, you can look for them around there. The hex code for SPACE is >20, for ENTER it's >0D.

If you want to try to change other keys, here are the start addresses for each of the six tables:

16E0 Joystick codes
1700 Lower case
1730 SHIFT codes
1760 FCTN codes
1790 CTRL codes
17C0 Key scan units 1 and 2

To figure out what keys correspond to what codes in these tables, convert to decimal and compare to the charts in the TI Basic manual listing FCTN and CTRL keys.

Remember that all address are in GROM/GRAM, and you will need to enable bank 1 or 2 when making any changes.

A note about the lower case key scanning: when you have Alpha-Lock down, in the capitals position, the key scan routine reads the key code from the LOWER CASE table, NOT the SHIFT table. If the key is a letter (ASCII range 97-122) and the Alpha-Lock is down, the key scan subtracts ASCII 32 from the key code, which moves it from the lower case portion of the alphabet to the upper case portion. If, however, the Alpha-Lock is down AND you are pressing SHIFT, it gets the key code from the SHIFT table.

One final caution: a few programs include their own key scan routine, and as such, the don't scan GROM for the key code. Thus, the keyboard will revert back to normal when running these programs. Two programs that do this are MG Explorer and the GRAM Kracker Memory Editor. While these programs are few and far between, you should keep it in mind if considering any major changes to the keyboard (i.e. converting it to DVORAK). But you should not let this stop you from making minor changes, like adding printer control codes for (X)Basic.

A FEW NOTES ABOUT MYARC'S EXTENDED BASIC AND THE GRAM KRACKER
by Craig Miller (MG)

Quite a few people have asked us about the MYARC Extended Basic and its use in the Gram Kracker.

Part of the MYARC XB system is an 8K RAM Module and a new PROM for your 128K/512K RAM Disk Cards. The module only contains 8K of static RAM, it does not contain any programming. The new PROM that is installed in your RAM Disk has a power up routine that loads some information into this 8K Ram module every time you go back to the title screen.

If you want to use this XB with the Gram Kracker simply leave the Write Protect switch in the Bank 1 or Bank 2 position and then press RESET. This will allow the MYARC PROM to down load its information into that RAM Bank in the Gram Kracker, and appear on your menu. You MUST leave the Write Protect switch in the Bank 1 or 2 position in order for MYARC's XB to execute properly.

One thing to remember is, whatever was in the selected Ram Bank will be wiped out by the 128K/512K power up routine. (See the article on disabling the MYARC RAM-disk to fix this problem - MDD.) So if you had TI Extended Basic loaded into the Gram Kracker and you left the Write Protect switch turned off, then both XBs would appear on the menu BUT only the MYARC XB will work. TI Extended

Basic contains 2 banks of ROM and one of them will be wiped out so it will not execute properly.

There are a number of TI modules that do not contain any ROM they only contain GROM. As such these modules can properly reside in the Gram Kracker along with Myarc's XB. To find out if a module contains ROM simply plug it into the Gram Kracker's Module port and select 5 EDIT MEMORY. Next press FCTN = for HEX and set the address to C6000. If the memory window is full of 00 or FF, depending on your console, then that module only contains GROM. A few of the popular GROM only modules are, Editor/Assembler, TI-Writer, Disk Manager I & II, Multiplan and PRK. A few of the ROM/GROM or ROM only modules are TI Extended Basic, Mini Memory, Atari and most other third party modules.

DISABLING THE MYARC RAM-DISK POWER UP
by Mike Dodd

If you have the MYARC XBII cartridge, you have noticed that the RAM-disk always wipes out your ROM bank if you forget to enable the write protection. The following patch will disable the power up routine in the RAM-disk, which prevents it from clearing out your ROM bank. Now you can leave the write-protect off (e.g. to act as a Super-cart) and not worry about it being zapped!

To make the change, enter the GRAM Kracker Memory editor. Press FCTN 1 to select GRAM, and FCTN 5 for search. Type 0000 for the start, and 0300 for the end. Press FCTN 9 to enter the search window and type 8780D0. Press FCTN S to back the cursor onto the "0" in D0, and press ENTER. When it finds the string (mine was at g0183), press FCTN 5 to leave the search and FCTN 9 to enter the memory field. Write down the address it is at.

Now disable write protect and type 05190A. Press FCTN 9 again, use FCTN S to back over to the memory address, and type 190A. Press FCTN 9, ENTER to home the cursor, and type BF 80 D0 11 00 BF 80 D2 40 04 05. Now take the address you wrote down and add 3 to it (>0183 + >0003 = >0186). Type that address. Turn your write protect back on, press CTRL = to leave the editor, and re-save GROM 0 to disk. To save GROM 0, press 4 for Load/Save console, 3 for GROM 0, and 2 for Save console. Type the filename and press ENTER. Press space (the correct GROMs are already enabled), let it finish saving, and press space again. That's all there is to it!

If you wish to run MYARC XBII, disable your write protection, change switch 2 from GRAM 0 to Op Sys, and press reset. With GRAM 0 loaded, the patch is not in effect, so the MYARC RAM-disk will execute its normal power up routine. When the title screen appears, you can re-enable GRAM 0 and proceed as normal to load MYARC XBII.

Final note: if your RAM-disk is not backed up by an external power supply, you MUST run the power-up routine when you first turn the computer on. After that, if you reset the system you will not need to run the power-up routine again. ou have to run it the first time, otherwise the CALL PART and CALL EMDK commands will crash. To run it without it crahing your RAM bank, disable GRAM 0 (turn to Op Sys) when you turn on the computer, making sure that the write-protect is on. hen the title screen appears, enable GRAM 0 and don't worry about it again.

CHECKING THE W/P SWITCH IN XBASIC
by Mike Dodd

As you may have noticed, if you enter Extended Basic with the write protection off, your computer will lock up. If it doesn't immediatly, it will as soon as you type a command. This patch will make XB check the position of the write protect every time you enter XB. If the W/P is off, it will reset to the title screen and refuse to let you enter the cartridge.

To make the patch, load your GK

Utility I version of Extended Basic. Type G6372 for the memory address, FCTN = for hex mode, and FCTN 9 to enter the memory window. Enable bank 1 and type 06 D8 FB (don't type the spaces, they're just a guide). Press FCTN 9, back the cursor up over the memory address, and type D8FB. Press FCTN 9 and ENTER to home the cursor, and type:

```
86 A3 70 86 8F FC FA BD 00 8F ED 00
86 8F FC FC D5 00 8F ED 00 59 13 0B
00
```

Now, restore write protect, press CTRL = to leave the memory editor, and resave your cartridge to disk.

CHANGING THE XB "LIST" WIDTH
by Craig Miller (MG)

With Extended Basic loaded into the Gram Kracker you can change the LIST "device" width for your output device. This allows you to easily list your programs to printer in 28 columns, 132 columns or any width you choose. This same change will also change the DIS/VAR file width if you LIST to disk.

To make this change load Extended Basic into the Gram Kracker and then use the Gram Kracker's Edit Memory selection. Next press FCTN = for Hex, FCTN 1 for Gram Memory and FCTN 5 to activate the Search function. The Start address is 9000 and the Finish address is 9800. The Hex string to search for is: 00 12 00 00 00

When this is found press FCTN 5 to leave Search and FCTN 9 to put the cursor in the Memory Window. Turn on Bank 1 to disable Write Protection and move the cursor to the third 00 after 12 and change it to the width you would like (in Hex). In our XB this was found at g9170 and the byte to change was at g9174. Examples:

```
00 12 00 00 00 = default 80 column
00 12 00 00 1C = 28 column listings
00 12 00 00 84 = 132 column listings
00 12 00 00 FE = 254 column listings
```

The area you are changing is part of the default PAB for an Extended Basic LIST to a device. Since most of it is zeroed out it allows the card's DSR (i.e. RS232 or DSK) to set its own default for width. When you place a value here the card will use it instead of the default of 80 (>50).

If you want to LIST a 28 column program to disk and then load it into TI-Writer or the E/A Editor you will need to convert the file back into DIS/VAR 80 format. To do this simply run it through the following XB program, where TEST is a DIS/VAR 28 file and TESTA will be the DIS/VAR 80 file to be loaded into an editor.

```
100 OPEN #1:"DSK1.TEST",VARI
ABLE 28
110 OPEN #2:"DSK1.TESTA"
120 LINPUT #1:A$
130 PRINT #2:A$ :: PRINT A$
140 IF EOF(1)THEN CALL CLSAL
L ELSE 120
```

If the file is large you can easily convert it from DIS/VAR 28 to DIS/VAR 80 with a sector editor such as Advanced Diagnostics. To do this find the File's Header (File Descriptor Record) by doing a Find File. The "Sector" pointer at the top of AD's screen points to the File's Header Sector. Edit this sector and change the 17th byte, in hex, from 1C to 50 and then rewrite the sector. NOTE: This will only work if you are converting files to a longer logical record length, i.e. DIS/VAR 28 or DIS/VAR 40 into DIS/VAR 80. It won't work for longer to shorter, i.e. DIS/VAR 132 or DIS/VAR 254 into DIS/VAR 80

NOTE: if you are using the GK Utility I version of Extended Basic, you do not need to make this change, as included in the GK Utility patches are a method of setting the line width with the LIST command.

EXTENDED BASIC CALL INIT CORRECTION
by Craig Miller (MG)

Presently the CALL INIT loads >600 bytes starting at >2000 in Low Expansion Memory but only >4F3 bytes need to be moved. Because of this, some routines that were loaded into Low Expansion Memory get overwritten. The patch corrects this situation.

With EXTENDED BASIC loaded in the Gram Kracker, select 5 Memory Editor from the Gram Kracker menu.

Press FCTN = for HEX, FCTN 1 for Gram Memory Window and then FCTN 5 for SEARCH.

Type in C200 for the START address and C300 for the FINISH address. Press FCTN 9 to put the cursor in the Search String Input field and type in 31 06 00. Press FCTN S (left arrow) to place the cursor on top of the last byte to search for and press enter.

Turn off Write Protection, press FCTN 5 to leave SEARCH and press FCTN 9 to put the cursor in the Memory Window. Now replace 31 06 00 with 31 04 F3.

Restore write protect, return to Gram Kracker loader and resave module.

CALL INIT will now work "a little" quicker and it will not move unnecessary bytes out to Low Memory Expansion.

NOTE: if you are using the GK Utility I version of Extended Basic, you do not need to make this change, as it is included in the GK Utility patches.

CHANGING THE CURSOR SHAPE
by Mike Dodd

To change the cursor shape, in XBasic and Basic, search for 00 7C 7C 7C 7C 7C 7C 7C. With XBasic, search from g6000-7800. With Basic, search from g2000-3000 (remember to turn off the loader!). With the Editor/Assembler, search for 00 7E 42 42 42 42 7E 00 from g6000 to g7800, unless you've moved it elsewhere. If you moved it to Gram 7, search from gE000-gF800. Once you find it, change it to whatever shape you desire. It's the same format as in a CALL CHAR statement. Remember to turn off write protect before you change it and then turn it back on when you're done.

GK UTILITY 1 ENHANCEMENTS AND MODIFICATIONS
by Tom Freeman, LA 99ers

RETAIN GRAMS 1 AND 2 FOR YOUR OWN USE

Some users who have loaded Danny Michael's fine new combination Extended Basic and Editor/Assembler modules into their Gram Krackers may wish to preserve the use of TI-Writer at the same time. I had previously loaded GRAMs 1 and 2 with E/A and TI-W respectively, and thus this new program, which uses these two GRAMs to hold the ASSM1 and ASSM2 files for rapid loading, were no longer available. I had already modified these modules to load the files from my RAMdisk, which is also quite rapid, so I did not need Danny's rapid loader. However, I did wish to use the combination and make use of the other enhancements, such as cataloging from E/A and preserving file names.

The following modifications to your FINISHED files will accomplish the task. Essentially, I went to the area of Danny's code where the assembler was loaded from GRAM into CPU, and changed it back to the original E/A code, with some address changes because of the move to GRAM 7, and screen location changes. All the other routines

used by E/A to get the program from the disk were preserved.

To accomplish the changes, go to the GRAM KRACKER memory editor (press 5 on GK title screen), then FCTN 1 to get to GRAM memory, FCTN = to get to HEX, enter, and then type in E658. You should see in the memory window code beginning with the following bytes: 06 F4 60. Press FCTN 9 to replace the first three lines of code with the following (where you see ASCII text you can type in ASCII, which saves half the typing - also remember to push the W/P switch to Bank 1 or 2 while you are typing):

```
gE658 08 8B A1 14 4C 6F 61 64 20 41 73 73 '****Load Ass' '*+A*********'
gE664 65 6D 62 6C 65 72 28 59 2F 4E 29 3F 'embler(Y/N)?' '************'
gE670 20 FB 06 E7 9F D6 75 0F 60 5A D6 75 ' *****u*'Z*u' '*****?v*****v*'
gE67C 4E 60 5A 06 E5 B2 E6 28 06 E5-D4 BF 'N`Z****(****' '*****R*****t_'
```

Defaults for Assembler Source Code File

Danny's mods retain separate default areas in GRAM 2 for all the file or device names you input - only those for LOAD and SAVE file in the Editor are the same. I personally wish to have the last file name I used for SAVE in the Editor appear as the default for the Source Code in the Assembler, since I normally assemble source code I have just written and saved. This is easily done by positioning the cursor after the g in the upper left corner, typing F347, then FCTN 9 to get in the memory window. Replace the first byte 4C with 88 (W/P off!).

While you are making changes, you might consider the following:

1) if you are in fact loading the TI-W and E/A utility files from RAMdisk, then you should change the device name/number at gE61E (I use DSK4.) The length should still be 5 bytes.

2) I have also changed the name of the default program name for option 5 Run Program File from UTIL1 to another name. You can do this at gE62D (see article on changing drive defaults elswhere in Kracker Facts).

3) The format RAMdisk option from Danny's main E/A screen does not work if you have the RAMdisk with XBasic, because the CALL PART now requires three numbers rather than 2. To make sure you do not choose this option by mistake, go to gE0F8 and change the words "Format RAMdisk" to "Non-valid Key " and change the bytes at gE05A from 52 B1 to 40 5A. You will now stay on the menu screen if you hit 7.

BE SURE you have saved your original modified module BEFORE you make the changes. You should now save your newly modified module under a different name. GRAMs 1 and 2 will no longer be used for the ASSM files and you can go back to keeping other modules in this space, so long as the high bytes in GRAM 2 from 5ED4 to 5FFF are not used (Danny uses them to hold the default file names in E/A). Also note that because these 2 GRAMs in the GK are not used, Danny's mods are now also useful in the 56K version of GRAM KRACKER. However the default file names for E/A mentioned above will no longer work; you would always see garbage when you are prompted for a file name. It is easily eliminated with FCTN 3.

Using MSAVE

As there are still 2609 bytes of memory free at the top of the E/A in GRAM 7 (from >F5CE on) you could still store a few short Basic programs if you use the following (slightly cumbersome) method:

1) If you are using GRAM 2, save it using Option 4 Load/Save Console from the GK main menu. The third switch must be in the GRAM 1-2 position. Also save the "module" (Menu 2) since we will be clearing the module space. If you have a 56K GK without GRAMS 1-2 see NOTE below.

2) Move the entire contents of GRAM 7 to GRAM 2 (Gram memory - FCTN 1 until a g appears in the upper left corner if it isn't already there, E000 for Start, FFFF for Finish, g4000 for Dest, then FCTN 2 to move).

3) Initialize the module space (Menu 3).

4) Load module (Menu 1) with MSAVE from the original GK utility disk.

5) Go back to the Memory Editor (Menu 5), FCTN 1 to get to G memory, FCTN = for HEX. Press enter, then type in E012. In the memory window you should see E2 B7 E2 B7. Press FCTN 9 to get the cursor in there, then type F5 CE F5 CE (W/P off!). FCTN 9 again, move the cursor back over the memory address and change it to E1DD, FCTN 9 and change this E2 B7 to F5 CE also.

6) Move the 35 bytes at E2B7 to F5CE by entering E2B7 for Start, E2D9 for Finish, and gF5CE for Dest. Then FCTN 2 to move. Put Switch 4 back in W/P position.

This new MSAVE will save Basic programs starting at F5CE, rather than E2B7, leaving enough room for the E/A module. Save it with a new name (such as MSAVE plus your initials) with Menu 2

You may now go to Basic (GRAM 1-2 switch down and Loader OFF), enter your basic programs, and save them by entering CALL MSAVE. When you are done, and quit Basic, you should see them on the main console menu.

Now go back to the GRAM KRACKER, and save module again (using yet another name, just in case). You are now ready for your final modification of GRAM 7.

7) Go back to the GK Memory Editor, FCTN 1, FCTN =, and examine the 2 bytes at E012. This represents the first free address after your programs. Therefore you will want to save all the bytes from F5CE to that address.

8) Making sure that g is in the upper left corner, and 3rd switch is in GRAMS 1-2 position type in F5CE for Start, the bytes you just found for Finish, and g55CE for Dest, and press FCTN 2 to move.

9) The final change is at g4010. This is the address for the next application header after Editor/Assembler and must contain F5CE. Type it in.

10) Reload the module you saved in Step 1). 11) Move the entire modified contents of GRAM 2 to GRAM 7 by typing 4000 for Start,5FFF for Finish, gE000 for Dest and then press FCTN 2.

12) Save your new "module" with resident Basic programs under a new name. Remember that to USE these Basic programs the loader must be OFF, and switch 3 must be in TI Basic position.

NOTE: If you have a 56K GK, make the follwing changes in above steps:

1) You can't save GRAM 2

2) Move GRAM 7 to GRAM 3 by using g6000 for Dest. NOW clear everything else by a) Start 8000 Finish FFFF, W/P to Bank 1, FCTN 3 (FILL). b) FCTN 1 twice to get to CPU memory, Start 6000, Finish 7FFF, FCTN 3 c) switch W/P to bank 2 and hit FCTN 3 d) Save "module" (Menu 2) - this should give you one file on disk e) W/P ON (mid position).

3) to 7) are the same

8) First reload the "module" you saved in Step 2d). Then move the bytes with g75CE as Dest

9) The change is at g6010. BEFORE going to next step, a) Move GRAM 3 to GRAM 7 (Start 6000 Finish 7FFF Dest gE000, W/P to Bank 1, FCTN 2 b) Clear GRAM 3 (Start & Finish the same, FCTN 3) c) W/P ON (mid position) d) Save module - this will give GRAM 7 only.

10) is the same

11) Load the "module" saved in 9d)

12) is the same

All this is not as complicated as it sounds - I just detailed all the steps so you won't make any mistakes.

EXTENDED BASIC PROGRAM LOADER

program by Mike Dodd
technical information by Tom Freeman
article by Mike Dodd and Tom Freeman

I once asked Craig Miller whether it was possible to run XBasic programs directly off the menu, as MSAVE does with Basic programs. The answer was no, and essentially that is true, at least as far as having them run directly from GRAM is concerned, since the XML instruction needed exists only in Basic. But I kept on thinking that if XBasic can load a program called LOAD automatically from drive #1, why can't it do others as well! What follows is a program for doing this! The method involves the following concept: when XBasic starts up, it does a certain amount of housekeeping, and then inserts the string DSK1.LOAD into the crunch buffer in VDP ram, preceded by the length byte >0B and followed by byte >00, and then "pretends" that you typed it in with RUN, and runs it. It turns out that this area is never touched by the housekeeping chores, and hence can be done right at the start. Thus my method involves inserting the program name of your choice there instead, and setting up proper code to make an additional item on the menu. If the program isn't there, you get the same result as XBasic if LOAD isn't in drive 1 - just the "ready" prompt.

When you run the program, it first checks to see if the WRTGRM subroutine is loaded. If not, it attempts to load an object file called DSK1.WRTGRM/O (see article on writing to GRAM from XB elsewhere in Kracker Facts). After the routine is loaded, or if it is already loaded, the program presents a title screen and asks you to enter the start hex address to store the loaders. You should consult your GK Utility I manual for the locations of free space. A good place to store it is starting at hex B601 and continuing to B7FF, which is enough room for many loaders. If you are not using the GK Utility I version of Extended Basic, you can use 7800-7FFF, 9800-9FFF, or B800-BFFF, as these areas are free. Note that if you install the auto-load bypass patch into XB (see elsewhere in Kracker Facts), 7800-780A are used. After you enter the address, it will instruct you to enable bank 1 and press FCTN. Do so. It will then instruct you to restore the write protect switch and press FCTN. Again, do so.

Now it will ask you for the menu entries. The program will display the current hex address. You should be sure that it does not go past the last free address in your memory space. If it does, you should break the program and re-run it to avoid overwriting existing code in your cartridge. The computer will now ask you for the name to be placed on the menu. The name may not be more than 18 characters long, and it must be in all capital letters. It will then ask you for a filename (e.g. DSK1.MENU, DSKR.FWR, RD.XXB). Note that the filename can not be greater than 15 characters. After you enter the filename, the program will tell you if either of the entries are too long. After a short pause, the program will prompt you for another menu name and filename. When you are done entering all the loaders you wish to install, enter *** (three SHIFT 8s) for the menu name. The computer will then prompt you to enable bank 1 and press FCTN. Do so. The computer will now write the loaders out to the Extended Basic cartridge. After it is done it will prompt you to restore the write protect and press FCTN. After you press FCTN, the program will end. You may now type BYE, enter the GRAM Kracker Loader and save your modified cartridge.

By the way, after the GKXBLOAD program is a short program that I (T.F.) wrote allowing you to set up all your favorite programs to run without typing in the names: you merely insert them in the DATA statement, and follow the last with a "". If you save this program on your utility disk and create a menu entry for it with GKXBLOAD, you will quickly get a menu of these programs when you press the "MISC. PROGRAMS" key and be able to pick your program with one more key press. This way you can still have the auto load of DSK1.LOAD for use with programs that need it. For this program to run properly you MUST type in line 170 first, exactly as written!

```
100 DEF A(B)=B-65536*(B<0)::
 DEF A$(B)=CHR$(INT(A(B)/256
))&CHR$(B AND 255):: OPTION
BASE 1
110 ON ERROR 120 :: CALL LIN
K("WRTGRM"):: ON ERROR STOP
:: GOTO 130
120 CALL INIT :: CALL LOAD("
DSK1.WRTGRM/O")
130 DISPLAY AT(1,1)ERASE ALL
:"XBasic programs direct fro
m  the main menu": :"require
s GRAM Kracker (tm)"
140 DISPLAY AT(5,1):"Program
 by Mike Dodd": :"Technical
information by Tom  Freeman,
 LA 99ers"
150 DISPLAY AT(10,1):"Start
GROM address?" :: ACCEPT AT(
10,21):C$ :: CALL HD(C$,BG)
160 CALL LINK("WRTGRM",25554
,CHR$(149),25403,A$(BG+10)&A
$(BG),BG,"1"&CHR$(0)&CHR$(11
)&CHR$(168)&" cQ"&CHR$(5)&"c
r")
170 DIM B$(15):: E=0 :: CALL
 KEY(3,F,G):: H=BG+10
180 CALL DH(H,C$):: PRINT "(
now at ";C$;")" :: INPUT "Me
nu name? (*** to end)  ":
C$ :: IF C$="***" THEN 230 E
LSE INPUT "Filename? ":D$
190 B=LEN(C$):: C=LEN(D$)::
IF B>18 OR C>15 THEN PRINT "
ERROR - MAX LENGTH FOR MENU
NAME IS 18, MAX FOR FILENAME
IS 15" :: GOTO 180
200 E=E+1 :: B$(E)=A$(0)&A$(
H+7+B+C)&CHR$(B)&C$&CHR$(C)&
D$&CHR$(0)&"1"&CHR$(0)&CHR$(
C+2)&CHR$(168)&" "&A$(H+5+B)
&CHR$(5)&"cr"
210 IF E>1 THEN B$(E-1)=A$(H
)&SEG$(B$(E-1),3,255)
220 H=H+LEN(B$(E)):: IF E<15
 THEN 180
230 CALL SOUND(200,1200,0)::
 DISPLAY ERASE ALL
240 D=1 :: DIM E$(2):: E$(1)
,E$(2)="" :: FOR B=1 TO E ::
 IF LEN(E$(D))+LEN(B$(B))>25
5 THEN D=D+1 :: B=B-1 ELSE E
$(D)=E$(D)&B$(B)
250 NEXT B :: IF D=1 THEN CA
LL LINK("WRTGRM",BG+10,E$(1)
):: END
260 CALL LINK("WRTGRM",BG+10
,E$(1),BG+10+LEN(E$(1)),E$(2
)):: END
270 SUB HD(A$,A):: A=0 :: FO
R X=3 TO 0 STEP -1 :: A=A+16
^X*(POS("0123456789ABCDEF",S
EG$(A$,4-X,1),1)-1):: NEXT X
271 A=A+65536*(A>32767):: SU
BEND
280 SUB DH(B,A$):: T=B-65536
*(B<0):: A$=""
290 Q=INT(T/16):: R=T-16*Q ::
 A$=SEG$("0123456789ABCEF",
R+1,1)&A$ :: IF Q THEN T=Q ::
 GOTO 290
300 SUBEND
```

MENULOAD

```
100 DATA RD.PRO1,RD.PRO2,""
110 CALL CLEAR
120 X=X+1 :: READ A$(X):: IF
 A$(X)<>"" THEN 120
130 DISPLAY AT(1,1)BEEP:"PRE
SS FOR" :: FOR Y=1 TO X-1 ::
 DISPLAY AT(2*Y+1,2):Y;" ";A
$(Y):: NEXT Y
140 CALL KEY(0,K,S):: IF S=0
 THEN 140 ELSE K=K-48
150 CALL INIT :: B$=A$(K)::
L=LEN(B$):: CALL LOAD(-45,L+
4):: CALL LOAD(-42,L)
160 FOR X=1 TO L :: CALL LOA
D(X-42,ASC(SEG$(B$,X,1))):: 
NEXT X :: CALL LOAD(X-42,0)
170 RUN "0123456789ABCDEF"
```

A ROUTINE TO WRITE TO GRAM FROM XB
by Mike Dodd

Although the GK Util I version of Extended Basic includes a POKEG routine, it is not useful for programs to modify Extended Basic because of the fact that if you disable the write protection, XB will lock up. I have written an assembly subroutine for Extended Basic that prompts the user to enable and disable the write protection.

To use the WRTGRM subroutine, use the format:

CALL LINK("WRTGTM"[,address,str-var...])

In other words, you must specify a decimal address and a string containing the data to write. If you wanted to write a hex 00 01 02, you could use:

A$=CHR$(0)&CHR$(1)&CHR$(2)

The address must be from -32768 to +32767. If the address is greater than or equal to 32768 (hex >8000), you must subrtract 65536 from it (IF ADDR>=32768 THEN ADDR=ADDR-65536).

You may pass multiple data sets to the WRTGRM routine. If you wanted to write the data in A$ to GROM >2000 (decimal 8192) and the data in B$ to GROM >A000 (decimal 40960 - 65536 = -24576), you would use:

CALL LINK("WRTGRM",8192,A$,-24576,B$)

You can pass up to seven data sets in one CALL LINK this way.

You also have the option of not specifying any data - just a simple CALL LINK("WRTGRM"). This will not do anything, other than let your program verify that WRTGRM is present in memory. For instance:

```
100 ON ERROR 110 :: CALL LINK("WRTGRM"):: GOTO 120
110 CALL INIT :: CALL LOAD("DSK1.WRTGRM/O")
120 program continues...
```

When WRTGRM is executed, it first checks to see if any parameters were passed to it. If not, it returns to XB. If so, it displays on the screen (at row 13, column 5) a message prompting you to enable bank 1 and press FCTN. After you enable bank 1 (or two, it really doesn't matter), press the FCTN key. When it is done writing all the data passed to it (almost instantly), it will prompt the user to restore write protect and press FCTN. Move switch 4 back the the center (write-protect) position and press the FCTN key.

For an example of the use of WRTGRM, examine the listing of my GKXBLOAD program (article elsewhere in Kracker Facts).

Here is the source code to WRTGRM:

```
0001 * WRITE TO GRAM FROM EXTENDED BASIC
0002 *  COPYRIGHT 1987 BY MIKE DODD
0003 *   116 RICHARDS DRIVE
0004 *   OLIVER SPRINGS, TN 37840 USA
0005 *   615/435-1667
0006        DEF  WRTGRM
0007        IDT  'MIKEDODD'
0008 VWA    EQU  >8C02
0009 VWD    EQU  >8C00
0010 GWA    EQU  >9C02
0011 GRA    EQU  >9802
0012 GWD    EQU  >9C00
0013 NUMREF EQU  >200C
0014 STRREF EQU  >2014
0015 FAC    EQU  >834A
0016 HFF    BYTE >FF
0017 BANK1  TEXT 'Enable bank 1&press FCTN'
0018 BANK0  TEXT 'Restore W/P & press FCTN'
0019        EVEN
0020 * PRINT WITH BASIC OFFSET. IN: R0=VDP
0021 *  ADDR,R1=CPU ADDR OF TEXT,R2=LENGTH
0022 PBASIC DATA SUBWS1,PBAS1
0023 PBAS1  MOVB *R13,R0
0024        MOVB @1(R13),@VWA
0025        ORI  R0,>4000
0026        MOVB R0,@VWA
0027        MOV  @2(R13),R0
0028        MOV  @4(R13),R1
0029 PBAS2  MOVB *R0+,R2
0030        AI   R2,>6000
0031        MOVB R2,@VWA
0032        DEC  R1
0033        JNE  PBAS2
0034        RTWP
0035 WRTGRM LWPI MYWS
0036 * GET # PARAMETERS. IF 0, RETURN
0037        MOVB @>8312,R6
0038        JEQ  RETURN
0039        SRL  R6,9          TO LSBY & /2
0040 * PRINT "ENABLE BANK 1..."
0041        LI   R0,>184
0042        LI   R1,BANK1
0043        LI   R2,24
0044        BLWP @PBASIC
0045        CLR  R12
0046 * WAIT FOR FUNCTION KEY TO BE PRESSED
0047 FCTN1  TB   7
0048        JEQ  FCTN1
0049        CLR  R8
0050 A      CLR  R0            NOT AN ARRAY
0051        INC  R8
0052        MOV  R8,R1         PARAM. NUMBER
0053        BLWP @NUMREF       GET NUMBER
0054        LWPI >83E0         GPLWS
0055        BL   @>12B8        FLOATING->INT
0056        LWPI MYWS          MAIN WS
0057        MOV  @FAC,R9       GET ADDR
0058        CLR  R0            NOT AN ARRAY
0059        INC  R8            STRING
0060        MOV  R8,R1
0061        LI   R2,BYTESL     SPOT FOR DATA
0062        MOVB @HFF,*R2      255 CHARS
0063        BLWP @STRREF       GET IT
0064 * SAVE CURRENT GROM/GRAM ADDRESS
0065        MOVB @GRA,R7
0066        SWPB R7
0067        MOVB @GRA,R7
0068        SWPB R7
0069        DEC  R7            CORRECT
0070 * SET GRAM ADDRESS
0071        MOVB R9,@GWA
0072        SWPB R9
0073        MOVB R9,@GWA
0074        MOVB @BYTESL,R9    GET LENGTH
0075        SRL  R9,8          TO LSBY
0076        LI   R0,BYTES      START OF DATA
0077 B      MOVB *R0+,@GWD     WRITE TO GRAM
0078        DEC  R9            DONE?
```

```
0079        JNE  B              NO
0080 * RESTORE OLD GROM/GRAM ADDRESS
0081        MOVB R7,@GWA
0082        SWPB R7
0083        MOVB R7,@GWA
0084        DEC  R6             ALL OF 'EM?
0085        JNE  A              NO, MORE
0086 * WAIT TILL USER LETS GO OF FCTN
0087 FCTN2  TB   7
0088        JNE  FCTN2          STILL PRESSING
0089 * PRINT "RESTORE W/P..."
0090        LI   R0,>184
0091        LI   R1,BANK0
0092        LI   R2,24
0093        BLWP @PBASIC
0094 * WAIT FOR FCTN TO BE PRESSED
0095 FCTN3  TB   7
0096        JEQ  FCTN3
0097 * RETURN TO XB
0098 RETURN LWPI >83E0          GPLWS
0099        B    @>6A           GPL
0100 SUBWS1 BSS  >20            WS FOR PBASIC
0101 MYWS   BSS  >20            MAIN WS
0102 BYTESL BYTE 0              LENGTH
0103 BYTES  BSS  255            PLACE FOR DATA
0104        END
```

E/A-GRAMDSK INFORMATION
by Craig Miller (MG)

If you are using the E/AGRAMDSK version for your Editor/Assembler module in the Gram Kracker you can enhance the cursor reaction time with the following changes. First load your E/AGRAMDSK version of the E/A into the Gram Kracker and then use the Edit Memory selection to change the following two items, in hex:

1. Edit g7AA6, it currently contains 06 FF change it to 03 FF. This is part of the delay before a key goes into auto repeat.
2. Edit g7BBB, it currently contains 0A 00 change it to 00 01. This is a delay loop between keys.

With these two changes in place you will notice that the cursor now moves a little faster around the screen and that it goes into auto repeat a little faster. The cursor blink speed is determined by the byte value at g7AA9. It is currently 03, changing it to 01 blinks faster and 0F blinks real slow.

These items were found by using DISkASSEMBLER to disassemble the EDIT1 file. Once the file is disassembled you can find items in the E/AGRAMDSK version loaded into the Gram Kracker by adding >5804 to the address shown in the right hand column of DISkASSEMBLER's output. This will then be the gxxxx address for editing.

If you want to change the default Tabs for the E/A Editor they are located at g7ED6 and they are offset by minus one. The EOF marker that appears on the editor screen is located at g8018 through g803F. The text that appears on the Command Line, when you press FCTN 9, is located at g8614 through g8757. Have Fun!

CHANGING THE CURSOR OF THE E/A-GRAMDSK UTILITY
by Tom Freeman, LA 99ers

If you have used the E/AGRMDISK utility that came with the GK and installed the CHARA1 file, you may have noticed that instead of a true cursor on the editor and assembler option screens you get a little 1f. This is because the E/A uses character >1F for its cursor here, and CHARA1 hasn't defined it as a block. As TI-WRITER never actually uses it as far as I can tell, you can redefine it to whatever shape you wish. I put in a solid block cursor, although the E/A module uses a hollow block. The eight bytes in question are located as the last two of the first sector of CHARA1 and the first six of the 2nd sector (if you have already created the E/A GRAM disk files, these wind up being on the 25th and 26th sectors of the fourth file created. You should see (00 40) (4C 50 10 1C 10 10) in these two sectors. Change these all to 7E for a medium size block, or 3C for a narrow block, or (00 7E) (42 42 42 42 7E 00) for a hollow block.

While you're at it, if you don't like the arrow instead of a circumflex (caret), then go to the next sector and look at the 10th to 3rd bytes from the end. If these are 10 28 44 10 10 10 10

00, you can change the 10 's to 00 's and get a regular caret back.

CHANGING THE DEFAULT DRIVE
by Tom Freeman, LA 99ers

Many of you Kracker Hackers may still be working with a TI disk controller, but have a MYARC or New Horizons RAMdisk. Up until now you have had to put the EDIT1, EDITA1, EDITA2, ASSM1, ASSM2, FORMA1, FORMA2 in Drive 1 because the E/A and TI-WRITER modules insisted on it. Therefore, if you wanted fast loading from the RAMDISK, it had to be Drive 1, thus disabling your true floppy drive 1. The following sections will show you how to change these modules to make the defaults for drives 2-9, and allow you to keep using all your disk drives as usual.

To keep repetitions to a minimum, I will review the process of using the GK Editor here. First save your module to disk (if you haven't already) using option 2 on the GK main screen. Then remove the module and reload the file off the disk using option 1. Now choose 5, the Editor. The cursor will be in the upper left hand corner, over a small c (indicating CPU memory). FCTN 1 will switch you to a small g (for GRAM). Press enter and you can now type the appropriate addresses that will be described. FCTN 9 will put the cursor on the memory window, and you can now make changes (be sure that the W/P switch is up, to Bank 1, or changes will not be accepted). After the change is made, exit the Editor with CTRL =, exit the GK with FCTN = or FCTN 9 and test your changes. If they are OK, go back to the GK and re-save the module.

I am not sure if there are different versions of these modules out there, which might make the addresses slightly different. If so you can use the Search feature of the GK. After getting the GRAM window with FCTN 1, press enter twice to get to Start address, enter 6000, then A000 for the Finish address, press FCTN 5 to activate the Search, press FCTN = if you need to change from ASCII to hex or vice versa, press FCTN 9 to get the cursor into the search entry field, then type the string you want, MOVE THE CURSOR BACK TO THE LAST TYPED ENTRY, and press enter. The GK will find the first occurence for you and put the address in the upper left corner. To edit what you have found press FTCN 5 again, then FCTN 9 and you will have the cursor in the memory field.

EDITOR/ASSEMBLER

The default disk drive for loading the Editor, Assembler, and UTIL1 files is at g6621. [search for EDIT1 if this isn't exactly right and you don't see it on the screen] The default name UTIL1 is at g662D. The name length of these files (all the same) is at g661D (in hex of course), and equals 0A (i.e. DSK1.EDIT1 etc.) If you wish to have a different program name as your default for the Utility option it must still have 5 characters.

If you have installed the EDIT1 and ASSM1-2 programs in high GRAM using the E/AGRMDISK utility that came with the GK, then these names are not needed and you can change even the length of the Utility program, provided you change the length byte at g661D (be sure to add the 5 for DSK1.) Alternatively you can change the NAMES of the EDIT1 and ASSM1-2 files with a Disk Manager to make them correspond in length to the name of the UTIL1 type program. Then just type in the new names in GRAM, as well as the new length byte. There is no room for names longer than 5, but they can be shorter. They must BEGIN at the same location - the unused characters will be ignored. If you have chained a UTIL1 type program together with the module for automatic loading on powerup (uses FCTN X when saving the module, see your GK manual for instructions) then use a 4 character name for the module - this makes the additional files 5 characters. E.g. if the module name is UTIL then the utility programs can remain UTIL1 (and 2 if used). If you installed the Editor and Assembler programs in high GRAM then the numbers would have to be higher than 1-2. I named the module F, and used the high GRAM option, so my utility program had to be named F4 and F5, and I therefore used F4 for the default at g661D as above.

As you can see there are myriad possibilities - do it the way YOU like.

When you have done all this you are

ready to go. First save the new module of course!! Now set up your RAMDISK to whatever drive you have chosen as your default. Use some copy program to copy the module files plus the utility programs (and the EDIT1 etc if they are not in high GRAM) to the RAMDISK. Now when you enter the Editor or Assembler, you'll get them in a flash! Disk Manager 1000 V3.5 and the MYARC Disk Manger Supreme both support more than three drives. If you are using Disk Manager II, see my article on changing that cartridge to allow more than three drives.

TI-WRITER

This one is a bit easier, because the default utility program name is not picked out of GRAM as such, but is put up on the screen. Hence there is no need to worry about the length byte, as the program measures it once you press enter.

First, the default drive number is at g6763 (actually DSK1. is at g6760). If this address is not correct you can search for DSK1. but there seems to be one at 65A7 as well. I am not sure of what the function of this one is, but not changing it seemed to make no difference.

The name of the Utility program is at g6B27 in English. Change it to whatever you wish (probably the same as the one in the Editor/Assembler, if you have them on the same disk). The other 7 languages (!) are located at 6CD0, 6EBA, 70A1, 725B, 7469, 763B, and 6EBA. You can change them if you wish - I didn't bother since I don't use them. As a matter of fact, elsewhere in Kracker Facts are instructions by Mike Dodd on how to get rid of these altogether, which will be useful in the future, because I've heard a rumor that eventually we will be able to get TI-WRITER and E/A in one GRAM!

CHANGING DM2 TO ACCEPT NINE DRIVES
by Tom Freeman, LA 99ers

When TI originally wrote Disk Manager II, the only disk controller available was TI's, which would not accept more than three drives. So, TI didn't allow DM2 to accept a drive number of three or higher. But today, with MYARC's disk controller and RAM-disks, many people have systems with drives numbered higher than 3. This patch will allow you to change DM2 to allow 4, 5, or even 9 drives!

One would think that there is a single routine that checks for this. I worked through this one with Explorer and found a routine and changed it. But when I went back to the module, the higher numbers were only accepted in some places. I wound up doing a little bit of educated guessing. I am pretty sure that what is listed below will make it all work without messing up any routines.

First the changes to the routines. A hex 33 is picked out of GRAM each time; you can see this as an ASCII 3 as well. I found the following locations necessary to change (all in GRAM): 724D, 72C0, 63F4, 6426, 650C, 675D, 685D. All but the 2nd and 4th also have a small r before the 3, so you can use r3 for the search, if the addresses aren't right. Change all of these to 4, 5 or whatever number you wish.

Next use the search feature to look for (1-3). There are 2 locations for each language. Change these to the number you chose above. This doesn't affect the running of the module but looks neater.

EARLY LOGO LEARNING FUN FIX
by Craig Miller (MG)

The problem with the Early Logo Learning Fun cartridge is that it won't work with the CorComp disk controller card. The exact problem is that this module has an APPLICATION PROGRAM name length of 00. When the Corcomp DSR goes thru the modules looking for Application names for the menu it starts moving them out and then it decrements the name length counter. >00 decremented is >FF or 255 bytes. This is what causes the mess on the title screen.

To correct this simply SAVE the module out to disk using a TI or MYARC disk controller and then LOAD it back

into the Gram Kracker. Then select the Gram Kracker's MEMORY EDITOR and change the byte at g6047 to 01 and resave the module.

It will not appear on the Corcomp menu but you can press 3 to start the module or you can press the space bar twice and it will appear on the TI Menu.

We tried putting a standard header in it for the Corcomp menu but it messed up the TI and MYARC menus, so it wasn't a good universal fix.

VIDEO CHESS FILENAME ENTRY
by Mike Dodd, LA 99ers

The Video Chess cartridge's lack of ability to save to disk can often be frustrating. The following modification will allow you to specify any filename; disk, RAM-disk, cassette, and probably even hard disk.

Load the Video Chess module into the GK. Now enter the memory editor. Select GRAM/GROM with FCTN 1 and hex mode with FCTN =. Enter search mode with FCTN 5. Search between 6000 and 7800 for 31 00 0F AB E8 60 60. When you find it, exit the search with FCTN 5, get into the memory window with FCTN 9, and type 06 78 00 05 72 65 (be sure to enable bank 1). Now press FCTN 5 to search. Leave the addresses alone, and search for the same string, by putting the cursor on the last "0" in the last "60", and pressing ENTER. Press FCTN 5 to leave the search, FCTN 9 and ENTER to home the cursor in the memory window, and type 06 78 00 05 72 FC. Press FCTN 9 and change the address field (upper left corner) to 7800. Press FCTN 9 and ENTER. Now type the following data (don't type the addresses - they're just a guide).

```
31 00 0F AB E8 60 60 08 FC 20 FE 00
FF 02 08 46 49 4C 45 4E 41 4D 45 3F
A0 FF 02 5F 20 FB BF 00 00 22 BE B0
00 4A 03 58 22 D6 75 0D 78 3E D6 75
07 78 00 A2 75 20 BC B0 00 75 91 00
58 22 BD 02 00 A7 02 00 22 78 00 34
02 AB F2 A0 22 BC AB F1 03 BF 00 0B
F2 A6 B0 00 20 91 00 93 02 58 55 00
```

Now enable the write protect, press CTRL = to leave the memory editor, and save your modified Video Chess cartridge to disk.

Now, whenever you tell Video Chess to save or load a file, it will ask for a filename. Press FCTN 3 to erase the filename if you make a mistake. None of the other FCTN keys (i.e. FCTN S, D, 2, 1) will work.

TIW-MOVER FIX
by Craig Miller (MG)

IF you use the TIW-MOVER utility to move TI-Writer to another Gram chip you will need to patch the FORMA1 disk file. This file currently contains a simple module check that won't allow it to run with the "5 RUN PROGRAM" option of E/A or ANY OTHER module loaded into Gram 3 (>6000->7FFF) that contains Basic Subprograms (CALLs), such as Extended Basic. To correct this you need to use a sector editor such as Advanced Diagnostics.

Once Advanced Diags is loaded place your TI-Writer disk in drive 1 with the write protect tab removed. Execute the AD command "FF FORMA1" to get the file information and the Start Sector. Once you have the Start Sector # (ss#) execute the AD command "ES ss#". At the 34th and 35th byte in the first data sector (start sector) of the file you will find the Hex value of 1000, change this to 1011. The 1000 is a NOP (no operation) the 1011 is a JUMP to >2040 which bypasses the module check and wipe out routine. After you have patched this word press FCTN 9 and then execute "WS ss#" to rewrite the sector. We hope this clears up any problems you may have encountered with the new utilities.

REMOVING FOREIGN LANGUAGE OPTIONS FROM TI-WRITER & DM2

by Mike Dodd

To remove the extra languages from the TI-Writer cartridge, load TI-Writer into the GRAM Kracker and select 5 Edit Memory. Type G6006 to select GRAM address >6006, and FCTN = to select hex mode. Press FCTN 9 to enter the memory window, enable bank 1 and type 60CB. Enable write protect, press CTRL = to leave the memory editor, and re-save your TI-Writer cartridge back to disk. That's all there is to it!

To eliminate the three extra languages from TI Disk Manager II, enter the memory editor. Type G8007 FCTN = FCTN 9. Enable bank 1 and type 5B. Enable write protect, press CTRL = to leave the memory editor, and re-save your DM2 cartridge back to disk. That's all there is to it!

GRAM PACKER HINTS

by Tom Freeman, LA 99ers

Several modifications have to be made to your operating system in GRAM 0 in order to make full use of the GRAM PACKER (written by J. Peter Hoddie, available from Genial Computerware). You will be using your Gram Kracker Editor to accomplish these (option #5 from the GK main menu). Rather than describe all the keystrokes each time, I will remind you of the general method here. First of all, when you get to the editor screen, press FCTN 1 once to get to GRAM memory. Now when you are instructred to search for a string, press FCTN 5 for search. The cursor will be on the "start" address. Accept the default of 0000 if it is there, or type it in, then press enter to get to "finish" and type 2000. Now press FCTN 9 and type in your search string, remembering FCTN = to get to hex if that is what you are searching for (in general it will be). Back up the cursor one space to get it over the last character in the search string then press enter. If the string is not found, the edit field will not change - if it is found, the address in the upper left hand corner will reflect the location of the first byte of the string found. Now press FCTN 5 again to get out of SEARCH, then FCTN 9 to edit, and type in the appropriate changes. You will need WP off in order to type in the changes - remember to turn it back on when you are finished typing.

The following set of changes need to be made only if you will wind up with more than 9 items on your main menu. There would be two problems if the changes were not made: 1) you wouldn't see any after 9 because of the double spacing! and 2) even if you could the key presses would be : ; < = > etc. some of which would actually be two keys (SHIFT and key). We will therefore enable single spacing on the main menu (thanks to Craig Miller in The Smart Programmer for this information) and change the sequence of key presses from numbers beginning with 1 to letters beginning with A.

First, to change to double spacing: Search for (hex) A3 52 00 3A. In many consoles this will be at 02E0. Change the 3A to 1A. Next comes a problem of another routine using temporary storage where we will need it (not actually involved with the double spacing, but needed if there ARE more than 9 items for the menu). Leaving the start and finish addresses the same, get back to SEARCH by pressing FCTN 5, FCTN 9 and type in 00 02 28 60 for the search string. You should find it at about 0380. FCTN 5 to get back to the memory window. The top line should read:

00 02 28 60 00 D6 28 AA 43 95 D2 29

change the 3rd, 7th, and 12th bytes:

00 02 40 60 00 D6 40 AA 43 95 D2 41

You should also insert the small capital character set into the TITLE SCREEN Characters using the NEWCHARS program on the original GK utility disk, otherwise the characters will touch each other top to bottom and be almost impossible to read. Note that you can only have 16 items on the menu because the start address for the first item is destroyed by the 17th item.

Now to change the key presses to letters - this is simpler. First change your start address back to 0000, then

search for BE 58 30. You should find it at about 0275. Change the 30 to 40. Next search for A6 75 31 (should be at 02FC) and change the 31 to 41. You will now see letters instead of numbers on the main menu.

I found another problem with many programs: they do not bother to change the keyboard unit to be scanned, assuming it to be 5, since that is where the E/A module is when option #5 is chosen. The problem is that the operating system is using keyboard unit 3 at the time the menu is set up (for this reason you can use lower case letters for the key press on the menu - they will be converted to upper case). Here is a simple fix: 12 bytes past the 41 you just typed in you should see 06 03 Cx A4, where the x is probably a E. Change the first three to 05 19 00. Now FCTN 9 to get out of memory window, move the cursor to the address after the g in the upper left hand corner, and type in 1900. Now FCTN 9 again, press enter to "home" the cursor, and type in the following: 06 03 Cx BE 80 C6 02 05 03 0y where x is the same as you just found above, and y is 3 higher (in hex) than the address where you found the 06 03 (if that was 030A as it was in my console, then y would be D). This changes the keyboard unit to 5.

For those of you using SBUG6, as I do often, and who wish to use it from the main menu, you may have found that the small character set is not loaded, which is a PAIN! It's OK if you have loaded it from E/A #5. Here is a fix: it incorporates MG's GPLLNK inserted directly into memory and then a simple BLWP @GPLLNK DATA >004A and then return to the beginning of the actual program. You will need a sector editor for this. First find the FDR of the file (catalog sector). Change byte 16(>10) from 92 to EA. Second find the first actual data sector of the file. Change byte 3 from 92 to EA and bytes 24-25(>18-19) from 62 84 to 7D D2. Finally go to the LAST sector of the file (there are 30 data sectors) and starting at byte 146(>92) carefully type in the following over whatever is there:

```
7D 7E 7D 9E 7D C2 17 6C 00 50 00 00
00 00 00 00 00 00 C8 1B 83 E8 C8 3E
83 EC C3 20 20 0E C8 09 20 0E 02 E0
83 E0 06 94 C9 20 7D 92 83 02 05 E0
83 73 04 60 00 60 C1 20 16 6C 06 94
02 E0 7D 7E C8 0C 20 0E 03 80 02 00
0B 00 C8 00 83 4A 04 20 7D 8C 00 4A
04 60 62 84
```

GRAM PACKER AID
by Mike Dodd, LA 99ers

One of the problems with GRAM Packer was that it has to know whether or not it the program uses TI Save format. Now you can use an XB program I wrote to analyze a file and tell you what format it uses, whereas before, the only way to tell was trial and error. Since my program must read sectors off the disk, you must load Barry Traver's RAW program before running the XB program. RAW was on Genial TRAVelER V1#4, and is present in all versions of the TRAVelER's XXB program.

When the program runs, it will ask for a filename. It will then analyze the file and tell you if it uses TI Save, doesn't use TI Save, or if it isn't an EA5 file. It may take a while, depending on the number of files on the disk, since it is written in XB.

```
100 !**********************
110 !*GRAM Packer utility*
120 !*Determines if file*
130 !*is TI-Save type or*
140 !*non - TI-Save type*
150 !*By Mike Dodd. Uses*
160 !*Barry Traver's RAW*
170 !*program.          *
180 !**********************
190 GOTO 200 :: A$,B$,C$,D$,
E$ :: A,B,C,D,E,F,G :: CALL
LINK :: !@P-
200 DISPLAY ERASE ALL:"MAKE
SURE BARRY TRAVER'S    RAW P
ROGRAM IS LOADED. IF   NOT,
PRESS FCTN 4 AND LOAD IT"
210 G=256 :: INPUT "FILENAME
? DSK":A$ :: A=VAL(SEG$(A$,1
,1)):: A$=SEG$(A$,3,10):: A$
=A$&RPT$(" ",10-LEN(A$))
220 CALL LINK("READ",A,1,B$,
C$):: B$=B$&SEG$(C$,1,127)::
 FOR E=0 TO 126 :: F=ASC(SEG
$(B$,E*2+1,1))*G+ASC(SEG$(B$
,E*2+2,1))
230 CALL LINK("READ",A,F,C$,
D$):: IF SEG$(C$,1,10)=A$ TH
EN 250
240 NEXT E :: PRINT "ERROR -
 NOT FOUND" :: END
250 D=(15 AND ASC(SEG$(C$,30
,1)))*G+ASC(SEG$(C$,29,1))::
 CALL LINK("READ",A,D,D$,E$)
260 E=ASC(SEG$(D$,1,1))*G+AS
C(SEG$(D$,2,1)):: IF E<>6553
5 AND E<>0 AND E<>887 THEN P
RINT "ERROR - NOT E/A 5 TYPE
 FILE" :: END
270 B=G*ASC(SEG$(D$,3,1))+AS
C(SEG$(D$,4,1))
280 E=ASC(SEG$(C$,17,1)):: F
=ASC(SEG$(C$,16,1)):: IF E=0
 THEN C=F*G ELSE C=F*G+E-G
290 IF B=C THEN PRINT "TI SA
VE" ELSE PRINT "NON TI SAVE"
```