MID-ILLINOIS
COMPUTER RESOURCE ORGANIZATION
P. O. BOX 766
BLOOMINGTON, ILLINOIS 61701-0766

MICRO/99 Newsletter
Volume 5, Number 1
January - February 1987

MICRO/99 is a not-for-profit group dedicated to the sharing of information and public domain software for the Texas Instruments 99/4A home computer. Members have free access to our library of several hundred programs on cassette and diskette. Meetings are held at 7:00 p.m. on the third Thursday of each month at the Illinois Agriculture Association building, 1701 Towanda Avenue, Bloomington. Attendees sign in with the guard at employee entrance number 4 at the rear of the building. Turn left at the sign for the main reception area and go down the stairs on the far side of it. Visitors are especially welcome, and may attend one meeting free of charge. Annual dues are $15 per family.

*** MEETINGS: JANUARY 15 & FEBRUARY 19, 1987 ***

At the January 15, 1987 meeting, Brian McFeeters will demonstrate a program for combining graphics and TI-Writer. He also plans to show Multiplan templates for income tax preparation. Yes, it is getting to be that time again!

By the February 19 meeting, Sid Smart will try to learn and share something about PILOT/99. PILOT stands for Programmed Inquiry Learning Or Teaching. It is a language particularly well suited for computer aided instruction. (I've intended to do this for some time. Now I'll have to!)

**** 1987 OFFICERS ****

After a long, hard campaign, the incumbents have beaten off all challengers and retained their offices for another year! In case you can't tell who the figureheads are in this loosely structured group, we supply the following list.

President (and assistant editor):
Sid Smart, 804 E. Cherry, LeRoy, IL 61752 (309) 962-9305

Vice President (and newsletter editor):
Brian McFeeters, 751 Pierce, Morton, IL 61550 (309) 263-8807

Treasurer:
Ray Hinrichsen, 407 Hilltop Ct, Bloomington, IL 61701 (309) 827-6965

Software Librarian:
Aubrey Johnson, 510 McKinley, Normal, IL 61761 (309) 452-3210

Newsletter production and distribution:
Ray Fisher, 2704 Sheffield, Bloomington, IL 61701 (309) 663-2745

**** SMART REMARKS ****

The new year is here. With the holiday activities behind us and long, cold nights upon us, most of us find more time to spend enjoying our great little TI/99. For owners of an orphan computer, the resources available to us through a users group are particularly important. The software libraries, newsletters, and personal contacts provided by the groups around the world form an amazing network that have kept our orphans not only alive, but thriving. That's my lead in to remind you that annual membership dues ($15 per family - what a bargain) are now due! Anyone who hasn't received $15 worth of software and information from the group in the last year really hasn't tried.

Sid Smart, President

Assembler Executing

by

Jim Lohmeyer

Hello from your friendly assembler programmer! I have decided to start writing some (a series?) of assembly language articles designed to help clear up some common misconceptions of assembler; like "I can't program assembly! I have a hard enough time programming my microwave to cook dinner." The only prerequisites for using this article are: 1. a TI99/4A computer 2. an Editor Assembler cartridge 3. at least 32K memory 4. at least one disk drive 5. a knowlege of how to create graphics in basic or XB and 6. a knowlege of how to use the EA. There are two separate parts to this article. The first is on bit-mapped graphics mode, the second is on using the SAVE utility.

This first article is for a moderate level assembly programmer. If anyone would like, I can do some basic level articles. Just ask.

PART ONE

BIT-MAPPED MODE EXPLAINED
or
HOW TO COMMIT HARI KARI WITH AN EA CART

Let's go back in time for a moment to the old ENTHUSIAST 99 magazine (published by Charlie Lafara's IUG). In the March 1984 issue their excellent assembly columnist Bill Gronos explained (finally) the ever mysterious bit map graphics mode. In his article Bill wondered how many people could still be sane after the EA manual's explanation of bit map mode (BMM). He then proceeded to explain BMM with an analogy using egg cartons, paint, t.v. cameras, and the two fictional characters Godfrey Grafix and Bob Bitmappe. After reading this article, at the tender age of 14 I was ready to (and did) give up assembly programming for about three months. I was thoroughly confused and really didn't care if Godfrey and Bob were eaten by a giant program bug. I am not criticising Mr. Gronos' article. Maybe my young mind was just easily confused or maybe at that point I was just looking for an excuse for a vacation. Maybe both.

In any event, the BMM is fairly easy to use. Before continuing, read the explanation of BMM in the EA manual (pp. 334-337). There. That wasn't so bad, now was it? Confused? If you are, then go back and re-read the part that gave you the most trouble. Believe it or not, all of the information that you need to use BMM is in those 4 pages, and in my opinion, presented rather well.

BMM has it's advantages and disadvantages. On the pro side: you can define up to 3 sets of 256 DIFFERENT characters, Each character can use all of the 16 colors (two per dot row), and sprites can be used (unlike in text mode). On the con side: although you can use sprites, their automatic motion feature can't be used. Also beause of the amount of graphic power available, BMM uses over 12K of the available 16K of VDP memory,

which, for one thing, reduces the amount of free VDP memory for sound tables.

AN OVERVIEW OF THE THREE VDP TABLES

There are three tables in VDP RAM that are used for screen access. They are: the Screen Image Table (SIT), the Pattern Descriptor Table (PDT) and the Color Table (CT). Each of the three tables is divided into thirds. Each third of each table is usable only with the same third of the other two tables. For example, the second third of the SIT is only usable with the second third of the other two tables. The computer automatically assumes this fact. Unfortunately, there is not a way around this.

The SIT

The Screen Image Table is 768 bytes long (one byte for each screen position). Each byte in the SIT contains a number from >00 to >FF (0-256). This table has 3 sections of 256 bytes each (256*3=768). The first section describes the first third of the screen (rows 1-8). The second third is for rows 9-16, and the last third is for rows 17-24. The character number placed in this table points to the proper character in the PDT. (REMEMBER: whatever third of the SIT you are working with is described by the SAME third of the PDT.) Each 32 byte block (starting from byte 0) will describe each line of the screen. The SIT is normally placed at address >1800 by setting VDP Register 2 to >06

The PDT

The Pattern Descriptor Table is 6194 bytes long (eight bytes for each character pattern). This table has three 2048 byte segments containing 256 separate character definitions each. The CHAR subprogram description in the USERS REFERENCE GUIDE (pp. II-76 - II-79) discusses the creation of character patterns. The patterns in the first third of the PDT describe ONLY the characters for the first eight lines of the screen. The second third describes characters for lines 9-16 ONLY and the last third describes characters for lines 17-24 ONLY. If, for instance, you wanted to write a program to put text on the screen, you would have to define the alphabet, and place it at the beginning of each third of the table. Conversely, you could define the alphabet in the last third only and use the other two thirds of the screen for graphics. Since these tables are set up this way, you could theoretically make each of the 768 characters on the screen be DIFFERENT! Quite a change from XB graphics, isn't it? The pattern descriptor table MUST be placed at either >0000 or >2000 in memory by placing >03 or >07 in VDP Register 4. If it is placed at >0000, the Color Table must be at >2000, and vice versa. When I am using BMM, I usually Place it at >0000, with the CT at >2000.

The CT

The Color Table describes the colors of the characters in the PDT. It is also 6194 bytes long (eight bytes for each color on a

character's dot row). This table is much simpler to use than many people think it is. It is also divided into thirds, with each third operating in the same manner as the previous two tables. Each entry in this table is in the form of eight bytes. Each byte describes the color of one of the eight dot rows of the character. Now is when most people start to panic. Each of those bytes is divided into halves with each half called a nybble. The first nybble of the byte is the color of the pixels that are ON in this dot row. The second nybble is the color of the pixels that are OFF in this dot row. So, with some simple multiplication, we find that each character can have 16 colors in BMM as opposed to only 2 as in XB's graphic mode. Quite a difference! The color codes are in the EA manual (p. 330). The color table MUST be at either address >0000 or >2000, and is placed there by putting either >7F or >FF in VDP Register 3.

COMMENTS AND RAMBLINGS

I would suggest that you study the examples on pp.336-337 for an even better understanding of the tables. The pixel-calculating program segment on p.336 is not only very useful, but also self explanatory.

PROGRAM DESCRIPTION

The program listing is a short program to demonstrate BMM. The source is completely commented, so it is pretty much self-explanatory. The character used, and the first set of colors are from the EA manual (p. 337). The major steps to using BMM mode in this program are:

1. REF/DEFS
2. set up workspace
3. set BMM
4. set up table addresses
5. clear tables
6. load tables with desired data
7. loop so screen is visible and QUIT key is active

The program is mainly to demonstrate the principle that the three sections of each table are separate and how they interact with each other. It also demonstrates the procedure of setting up the tables. The program in itself is really not something to write home about, but I think it serves its purpose. One thing to remember is that for really elaborate graphics you will create MANY long DATA statements. But it's well worth it.

In the next newsletter, we will take the source code from this article and convert it to program image using the SAVE utility. A small tutorial on just how that utility works will also accompany it. Until then, have fun playing with bit map graphics, and remember: the EA manual can answer almost ALL of your questions, sometimes you just have to LOOK HARD.

Happy assembling,
Jim

```
* SOURCE CODE FOR BIT MAP MODE DEMO
* FILLS SCREEN WITH 3 SETS OF 3 COLOR
* CHARACTERS
*
* MICRO 99 NEWSLETTER JAN. - FEB. '87
*
* SOURCE CODE BY JIM LOHMEYER 1/11/87
*
*
       REF VSBW,VMBW      * VDP UTILITIES
       REF VWTR
*
*  INITIALIZATION
*
MYREGS BSS 32             * REGISTERS
*
START  LWPI MYREGS        * LOAD MY REGISTERS
       LI R0,>0002        * SET BIT MAP
       BLWP @VWTR
       LI R0,>0403        * PDT @0
       BLWP @VWTR
       LI R0,>0206        * SIT @1800
       BLWP @VWTR
       LI R0,>03FF        * COLOR TABLE @2000
       BLWP @VWTR
       LI R0,>0706        * SCREEN COLOR
       BLWP @VWTR
*
*  INIT SIT
*
       LI R0,>1800        * ADDRESS OF SIT
       LI R1,>00          * CHARACTER 0 IN ALL POSITIONS
LOOP1  BLWP @VSBW         * WRITE IT TO VDP
       INC R0             * INCREMENT ADDRESS
       CI R0,>1800+768    * END OF SIT?
       JNE LOOP1          * NO, GO AGAIN
*
*  CLEAR PDT
*
       CLR R0             * PDT ADDRESS >0
       CLR R1             * VALUE=0
LOOP2  BLWP @VSBW         * WRITE TO VDP
       INC R0             * INCREMENT ADDRESS
       CI R0,>1800        * IS IT THE END?
       JNE LOOP2          * NO, CONTINUE
*
*  CLEAR COLOR TABLE
*
       LI R0,>2000        * ADDRESS OF COLOR TABLE
       CLR R1             * VALUE=0
LOOP3  BLWP @VSBW
       INC R0             * INCREMENT ADDRESS
       CI R0,>3800        * IS IT THE END?
       JNE LOOP3
*
       CLR R0             * ADDRESS OF PDT (1ST THIRD)
       LI R1,MON1P        * CHARACTER PATTERN
       LI R2,8            * 8 BYTES TO WRITE
       BLWP @VMBW         * WRITE TO VDP
       LI R0,2048         * SECOND THIRD OF PDT
       BLWP @VMBW         * WRITE IT
       LI R0,4096         * LAST THIRD OF PDT
       BLWP @VMBW         * WRITE IT
       LI R0,>2000        * ADDRESS OF COLOR TABLE (1ST THIRD)
       LI R1,MON1C        * COLOR OF CHARACTER 1 IN FIRST THIRD OF SCREEN
       BLWP @VMBW         * WRITE IT
       LI R0,>2000+2048   * SECOND THIRD OF COLOR TABLE
       LI R1,MON2C        * COLOR OF CHARACTER IN SECOND THIRD OF SCREEN
       BLWP @VMBW         * WRITE IT
       LI R0,>2000+4096   * LAST THIRD OF COLOR TABLE
       LI R1,MON3C        * COLOR OF CHARACTER IN LAST THIRD OF SCREEN
       BLWP @VMBW         * WRITE IT
ENDLP  LIMI 2             * ENABLE INTERRUPTS FOR QUIT KEY
       LIMI 0             * DISABLE INTERRUPTS
       JMP ENDLP          * LOOP THROUGH
MON1P  DATA >FF99,>99FF,>1924,>42C3  * PATTERN CODE FOR CHARACTER
MON1C  DATA >4646,>4646,>4D4D,>4D4D  * COLOR CODE FOR CHARACTER 1
MON2C  DATA >1E1E,>1E1E,>1D1D,>1D1D  * COLOR CODE FOR CHARACTER 2
MON3C  DATA >E1E1,>E1E1,>EDED,>EDED  * COLOR CODE FOR CHARACTER 3
       END START          * MAKE IT AUTO START
```

ARTCONVERT

by BRIAN MCFEETERS

As you can see, this article is not being printed in normal print style. I am using different print fonts converted for use with TI-WRITER or FUNWRITER. It is actually very easy to use.

The program I used to convert the different fonts is *ARTCONVERT* from *TRIO+ SOFTWARE*. It will convert small fonts in *TI-ARTIST* format that can be printed using *TI-WRITER* or *FUNLWRITER* formatter. The program disk comes with the script font and several instances. They can be converted for use on a *Prowriter* or *Epson* compatible printer. It can take up to *10* minutes for the conversion process, but they only need to be converted once.

The converted files are actually a series of transliterate commands (.TL) that when run thru the formatter either prints the instance or redefines the print style. Before your file can be run, all carriage returns must be removed. Also, a line feed must be added to every line to be printed.

Some of the scripts available from TI-ARTIST data disks are:

OFFBEAT
ROMAN
ROUND
SLANT
TECH2

Below is an example of an instance printed thru the formatter. Following that is a partial listing of the file used to create this article. All the fonts are addressed by include files (.IF DSK1.).

```
.CO ELONGATED PRINT ON (PROWRITER)
.e
.IF DSK2.OFFBEAT
          ARTCONVERT~
.IF DSK1.NORMAL
.CO ELONGATED OFF
.F
.IF DSK2.TECH2
             ~
                                  by BRIAN MCFEETERS  ~
                                  ~
.IF DSK1.NORMAL
.CO COMPRESSED AND ELONGATED ON
.bQ.e
.IF DSK1.SCRIPT
     As you can see, this article is not being printed in  ~
normal print style.  I am using different print fonts       ~
```

WINDMILL

The following program was written by Gary Cox of the Mid-South Users Group. It is a good sprite demo requiring the use of Extended Basic.

```
100 ! BOUNCING DANCING SPRITES
110 ! BY GARY COX (NOV86)
120 !
130 !Mid-South TI99/4A Users Group
140 ! Memphis, Tennessee
150 !
160 CALL CLEAR :: RANDOMIZE :: J=16 :: CALL SCREEN(2):: CALL CHAR(33,"00000000
00FFF00")
170 CALL CHARPAT(73,A$):: CALL CHARPAT(47,B$):: CALL CHARPAT(45,C$)
180 FOR I=1 TO 28 :: CALL SPRITE(#I,46,16,50,130,12,0):: FOR K=1 TO 30 :: NEXT
K ::  NEXT I
190 FOR I=1 TO 28 :: CALL MOTION(#I,4,0):: NEXT I
200 FOR I=1 TO 16 :: CALL COLOR(#I,I):: NEXT I
210 FOR I=12 TO 1 STEP -1 :: J=J+1 :: CALL COLOR(#J,I):: NEXT I
230 CALL COLOR(1,7,2):: CALL HCHAR(19,16,33,3)
240 FOR I=1 TO 28 :: A=INT(RND):: CALL MOTION(#I,-A,INT(RND)):: CALL SOUND(10,
(A+10)0,2,300,2,1000,2):: NEXT I
250 FOR I=1 TO 26
260 CALL CHAR(46,A$):: CALL CHAR(46,B$):: CALL CHAR(46,C$):: CALL CHAR(46,B$)
270 CALL DELSPRITE(#I):: CALL SOUND(100,-7,2)
280 NEXT I
```

HANDY TI TIPS

The following tips were collected by Rick Kellogg and appeared in the OCT86 newsletter of the CINDAY Users Group.

```
PROMPT 'BEEP'          CALL SOUND(150,1390,2)
PROMPT 'HONK'          CALL SOUND(70,218,1)
```

SPECIAL SCREEN CHARACTER CODES:

Slashed Zero	CALL CHAR(48,"0038444C54644438")	0
Right Arrow	CALL CHAR(??,"000804027F020408")	→
Left Arrow	CALL CHAR(??,"00102040FE402010")	←
Up Arrow	CALL CHAR(??,"081C2A4908080800")	↑
Down Arrow	CALL CHAR(??,"00080808492A1C08")	↓
Solid Line	CALL CHAR(48,"00FF")	—
Copyright Symbol	CALL CHAR(??,"003E415D515D413E")	©
PI Symbol	CALL CHAR(??,"0000FE2828282828")	π
Cent Mark	CALL CHAR(??,"00083C4848483C08")	¢
Check Mark	CALL CHAR(??,"0002020404482810")	✓

Note: For the above CALL CHAR's with ?? instead of a character number, you can add any number you are not using in your program.

Also, on some printers you can set the slashed zero as the default. On the Prowriter, dip switch SW2-1 should be closed for a slashed zero. Check your printer manual to see if you have that option.

MID ILLINOIS COMPUTER RESOURCE ORGANIZATION
P.O. BOX 766
Bloomington, IL 61701-0766

EDMONTON 99'ERS USER SOCIETY
P.O. BOX 11983, EDMONTON
ALBERTA, CANADA T5J-3L1

```
**************************************************************************
*        MMM   MMM   IIIIII    CCCCCCC   RRRRRRRR      00000000        *
*        MM M M MM     II      CC        RR     RR     00     00       *
*        MM M M MM     II      CC        RRRRRRRR      00     00       *
*        MM  M  MM     II      CC        RR     RR     00     00       *
*        MM     MM     II      CC        RR      RR    00     00       *
*        MM     MM   IIIIII    CCCCCCC   RR       RR   00000000        *
*                                                                       *
*                                                                       *
*        The MID ILLINOIS COMPUTER RESOURCE ORGANIZATION                *
**************************************************************************
```