VOLUME 2 ISSUE 3 $1.75 AUGUST 1986

"Serving 99'ers Since 1984"

THE SMART PROGRAMMER

Well, before we get started this month, please check your mailing label for the date of your subscription expiration. If you subscribed through Millers Graphics and have not yet renewed, you must renew by September 15 to ensure not missing a single issue.

Q&A

It is unfortunate that the Mini Memory Save and Load program in the July 1986 issue uses 17 sectors of a disk to store the contents of the module when, in some cases, it may only be necessary to save a small portion of the memory to be able to reconstruct the intended purpose of the program. Could it be modified in some way to use only the number of sectors necessary to retain the intended application?

This is quite an interesting question! As I pointed out in the article that accompanied the program, my intention was to allow quickly loading peek and poke values in the MM space without having to write a program that would have a REF/DEF table. The REF/DEF table, by the way, continues to grow with new entries, even in a new session (MM is battery-backed), until cleared. By loading the entire module RAM space, there is no possibility of pre-existing REF/DEF table entries remaining to use up memory or cause problems. But, for those who wish to save only a portion of the MM space, you might want to refer to this month's XB MIRROR program for an example of saving an Image of variable size. Still, AORG code that does not correct the FFAM could present a problem in identifying where code resides in MM. The Mini Memory Save and Load program uses the only safe universal save and load method I've found -- save the entire MM RAM area. It has also come to my attention that I published the program as soon as it worked and didn't correct a few comments and non-fatal errors. The TEXT for the PAB Filename used 16 spaces and should have used only 15, necessitating the addition of an EVEN directive. Under the BEGPAB label, hex was used instead of decimal, so the instruction should have read LI R2,25 instead of >25. The PABBUF EQU should have been commented as being the location of the PAB buffer, not its size. Finally, under the EXIT label, I omitted a register designation, which should have read MOV @SAVRTN,R11.

XB MIRROR

by Richard M. Mitchell

Here it is! New graphics capabilities for Extended BASIC! With the XB MIRROR program listed below, you can create a mirror image of an Extended BASIC (XB) screen and do a lot more. Before you get started, please note that you will need a GPLLNK and a DSRLNK, such as the ones we published

last month.

There are 18 new routines for you to use. The routines use four buffers in Low Memory to store VDP information. In the buffers, you can store copies of two Screen Image Tables (SIT's) and copies of two Pattern Descriptor Tables (PDT's) you have created. The SIT is the area of Video Display Processor RAM (VDP) that stores the characters that are on the screen (XB's GCHAR retrieves from the SIT). The PDT is the area of VDP in which the Hex codes that define the patterns of characters reside (XB's CHARPAT retrieves from the PDT). You can use these buffers through the routines provided by XB MIRROR to quickly "page" screens, regardless of whether you are using the mirroring routines or not. Refer to the XB demo and test program that follows the Assembly source code for examples of usage of the routines.

There are 2 routines to create a mirror image:

CALL LINK("MIRROR",A$,B$) uses a bit reversal routine to mirror individual characters in the range A$ to B$. A$ and B$ are single characters. For instance, if A$="A" and B$="Z", the patterns defined where the capital letters originally reside would be mirrored. B$ should be greater than or equal to A$ (refers to ASCII values). "MIRROR" alters the PDT, but does not affect any of the 4 buffers. WARNING: This routine does not do limit checking -- if A$ is greater than B$ or either is not in the valid XB ASCII range of 30 to 143, you may get undesirable occurrences.

CALL LINK("FLIP") "flips" the screen. In other words, the character at Row 1, Column 1 is replaced by the character at Row 1, Column 32 and the character at Row 1, Column 32 is replaced by the character at Row 1, Column 1, etc. To truly mirror a screen, you must use both "FLIP" and "MIRROR" (separating the functions provides increased versatility). "FLIP" alters the SIT. WARNING: This routine buffers through the first SIT buffer (SIT1), wiping out the previous contents of SIT1 (for speed and byte-efficiency).

There are 4 routines to put the contents of the VDP tables in the buffers in Low Memory:

```
CALL LINK("PUTSI1")
CALL LINK("PUTSI2")
CALL LINK("PUTPD1")
CALL LINK("PUTPD2")
```

There are 4 routines to get the contents of the buffers and place the data in the VDP tables:

```
CALL LINK("GETSI1")
CALL LINK("GETSI2")
CALL LINK("GETPD1")
CALL LINK("GETPD2")
```

There are 4 routines to save the contents of a buffer in Low Memory to a disk file, F$ (WARNING: The disk routines may not do adequate error checking. Some disk errors may go undetected.):

```
CALL LINK("SAVSI1",F$)
CALL LINK("SAVSI2",F$)
CALL LINK("SAVPD1",F$)
CALL LINK("SAVPD2",F$)
```

Finally, there are 4 routines to load the contents of a disk file to a corresponding buffer in Low Memory:

```
CALL LINK("LODSI1",F$)
CALL LINK("LODSI2",F$)
CALL LINK("LODPD1",F$)
CALL LINK("LODPD2",F$)
```

If you are experienced with Assembly Language, you should find it quite easy to extract individual routines to run on their own. Be sure to include all EQU's, DEF's, DATA's, BYTE's and BSS's that would be required. Once mirrored screens have been developed and saved to disk, your applications may require only the load and get routines (and possibly the put routines). As the program is presented, it uses 3,360 bytes for the 4 main buffers plus 186 bytes for the DSRLNK and GPLLNK (from last month) plus only 698 bytes for the 18 new routines, for a total of only 4,244 bytes including all buffers and workspaces! Note that this month's source code includes a COPY directive that assumes you have the DSRLNK and GPLLNK as 2 source files in drive 1.

After you have Assembled the source file, naming the object file "MIRROR/O", and have keyed in the XB program, copy the object file and the XB program onto a blank disk (so that you have a backup and your "work disk" has free space).

Next, run the XB program. The program will display a number in each corner of the screen and an arrow near the center of the screen and will then mirror the screen and proceed to test and demo all of the routines. Do not be concerned if you are using a television and cannot see the numbers at the edge of the screen -- a GCHAR will test the presence of the numbers, even if you cannot see them. As the program finishes, you should see the before and after (mirror) character patterns of ASCII 50 and 143 and the before and after values from GCHAR's of the 4 corners of the screen, as follows:

```
0038440810207C00
001C221008043E00
********************************
FEFCF8F0E0C08000
7F3F1F0F07030100
********************************
 49
 50
********************************
 50
 49
********************************
 51
 52
********************************
 52
 51
********************************
```

The 4 disk files for saving the buffers should occupy 4 sectors each for files "SIT1" and "SIT2" and 5 sectors each for "PDT1" and "1234567890" (the PDT2 file, which is named to test acceptance of the maximum filename length).

After running the program, some of you will probably be saying, "Wow!", but, unfortunately, some of you will find that you didn't key the program in properly. So, next month I'll cover the methods that I used to de-bug the program, in hopes that such a discussion might help you with both your own programs and ones you key in (no, I don't just "whip out" a program like XB MIRROR in an hour without errors -- it took me a lot of time and several de-bugging sessions).

The coding of XB MIRROR yields a couple of tips for beginners. First, multiplication can be accomplished with a bit shift. Under the label "MULPLY", SLA R1,3 was used to multiply the contents of Register 1 by 8. Each bit shifted advances by a power of 2. Shifting three bits is equivalent to multiplying by 2^3, 8. Of course, if any possible case would cause usable bits to be shifted out at the MSb (Most Significant bit, "left side"), a 32-bit MPY would be required. The second feature of XB MIRROR that is noteworthy for Assembly beginners is that the flow of the program does not jump around in a random, "spaghetti-code" fashion, but instead uses branches wisely to conserve bytes and make for readable code that was written from the top downward. The Top Down Method is a traditional, widely accepted and very useful programming technique that can (should?) be applied to most languages. The boundaries of routines are clearly discernable and common routines and values are shared efficiently.

For the few of you who haven't noticed by now, I really enjoyed writing XB MIRROR and I hope you enjoy using it. If interest dictates, I'll try to offer a CALL LOAD version in a future issue -- possibly a scaled-down version for Mini Memory, so that memory expansion and disk would not be required. As there is still plenty of memory available for the XB version, it will also be possible to add many more routines. Let me know what you would like to see added to the program! Maybe we could add a brief routine each month. I've already received several interesting suggestions, so it might be interesting to see how such a project might develop. You might also be interested to know that *Genial Traveler's* excellent XXB program AORG's into High Memory, so XB MIRROR and XXB (in Graphics, not Text mode) should currently be compatible!

```
**********************************
* XB MIRROR                      *
* BY RICHARD M. MITCHELL         *
* BYTEMASTER COMPUTER SERVICES   *
* COPYRIGHT 1986                 *
**********************************
        DEF   MIRROR,FLIP
        DEF   PUTSI1,PUTSI2,GETSI1,GETSI2
        DEF   PUTPD1,PUTPD2,GETPD1,GETPD2
        DEF   SAVSI1,SAVSI2,SAVPD1,SAVPD2
        DEF   LODSI1,LODSI2,LODPD1,LODPD2
PDT     EQU   >03F0           VDP ADDRESS OF BEGINNING OF PATTERN DESCRIPTOR TABLE
STRREF  EQU   >2014            STRING REFERENCE INTERFACE WITH XB
FAC     EQU   >834A            FLOATING POINT ACCUMULATOR ADDRESS USED BY GPL ROUTINE
STATUS  EQU   >837C            GPL STATUS BYTE (SEE BELOW)
WIPBEG  EQU   >8300            >8300 THRU >8340 ARE WIPED OUT BY GPL BIT REVERSAL
WIPEND  EQU   >8340            LAST WORD OF WIPE-OUT AREA
LEN1    BYTE  >01              LENGTH (ONE-BYTE CHARACTER FROM BASIC) (ALWAYS 1)
CHR1    BYTE  >00              ASCII VALUE OF FIRST CHARACTER FROM BASIC
LEN2    BYTE  >01              LENGTH OF SECOND CHARACTER FROM BASIC (ALWAYS 1)
CHR2    BYTE  >00              ASCII VALUE OF SECOND CHARACTER FROM BASIC
OFFSET  BYTE  >1E              REFERENCE TO BASIC'S FIRST 'REAL' CHARACTER, 30
        EVEN                   GET BACK ON EVEN WORD BOUNDARY
WIPBUF  BSS   >42              BUFFER TO STORE >8300 TO >8340 WIPED OUT BY GPL
WBUFEN  EQU   $-2              LAST WORD OF WIPE-OUT BUFFER
COUNT   BSS   2                BUFFER FOR NUMBER OF BYTES TO REVERSE
MYWS    BSS   >20              16 2-BYTE REGISTERS

MIRROR  LWPI  MYWS             SET OUR WORKSPACE POINTER
        CLR   R0               -
        LI    R1,1              | GET BEGINNING CHARACTER TO REVERSE FROM BASIC
        LI    R2,LEN1           |
        BLWP  @STRREF          -
        INC   R1                |
        LI    R2,LEN2           | GET ENDING CHARACTER TO REVERSE FROM BASIC
        BLWP  @STRREF          -
        CLR   R1               SET R1=0
        MOVB  @CHR1,R0         GET READY FOR CALCULATIONS
        MOVB  @CHR2,R1         GET READY FOR CALCULATIONS
        BL    @MULPLY          -
        AI    R1,8              | CALCULATE NUMBER OF BYTES TO REVERSE
        MOV   R1,@COUNT        -
        CLR   R0               SET R0=0
        MOVB  @OFFSET,R0       SET MSB OF R0=>1E
        CLR   R1               SET R1=0
        MOVB  @CHR1,R1         SET MSB OF R1=FIRST CHARACTER TO BE REVERSED
        BL    @MULPLY          CALCULATE BYTES INTO PDT
        LI    R0,PDT           SET R0=BEGINNING OF PDT
        A     R1,R0            ADD BYTES INTO PDT TO PDT TO SET FIRST BYTE TO REVERSE
        MOV   R0,@FAC          SET VDP ADDRESS OF FIRST CHARACTER TO REVERSE
*                                GPLLNK BIT REVERSAL USES FAC.
        MOV   @COUNT,@FAC+2     SET NO. OF BYTES TO REVERSE
*                                GPLLNK BIT REVERSAL USES FAC+2.
        CLR   R0               GET READY TO CLEAR THE GPL STATUS BYTE
        MOVB  R0,@STATUS       CLEAR THE GPL STATUS BYTE (MUST DO BEFORE GPLLNK)
        LI    R0,WIPBEG        -
        LI    R1,WIPBUF         | MOVE >8300 THRU >8340 TO BUFFER
SAVWIP  MOV   *R0+,*R1+         | MANY THANKS TO CRAIG MILLER FOR REMINDING TO DO THIS
        CI    R0,WIPEND         | IN THE MAY 1984 SMART PROGRAMMER.
        JNE   SAVWIP           -
```

```
        BLWP @GPLLNK      BRANCH TO GPLLNK AT POINTER INDICATED BY FOLLOWING DATA
        DATA >003B        POINTER TO GPLLNK BIT REVERSAL ROUTINE
        LI   R0,WIPBUF    -
        LI   R1,WIPBEG      | RESTORE WHAT GPL BIT REVERSAL WIPED OUT IN SCRATCH
RESWIP  MOV  *R0+,*R1+      | PAD RAM, >8300 THRU >8340.
        CI   R0,WBUFEN      |
        JNE  RESWIP       -
RETURN  LWPI GPLWS        LOAD WORKSPACE POINTER AT THE GPL WORKSPACE
        B    @>006A       BRANCH TO CLEAR STATUS BYTE AND RETURN TO BASIC.
*                           THANKS FOR THIS TIP IN GENIAL TRAVELLER, CREDITED TO
*                           PAUL CHARLTON

MULPLY  SB   R0,R1        SUBTRACT BYTES AND LEAVE DIFFERENCE IN MSB OF R1
        SWPB R1           SWAP BYTES TO SWITCH FROM BYTES TO WORDS
        SLA  R1,3         MULTIPLY BY 8
        RT                RETURN TO CALLER

VSBR    EQU  >2028        VIDEO SINGLE BYTE READ BLWP ADDRESS
VMBW    EQU  >2024        VIDEO MULTIPLE BYTE WRITE BLWP ADDRESS

FLIP    LWPI MYWS         SET WORKSPACE
        CLR  R2           SET R2=FIRST COLUMN OF CURRENT ROW
        CLR  R3           SET R3=OFFSET TO SIT1
ROW     MOV  R2,R0        SET R0=FIRST BYTE OF CURRENT ROW
        AI   R0,>1F       SET R0=LAST COLUMN OF CURRENT ROW
RC      BLWP @VSBR        READ BYTE FROM VDP
        MOVB R1,@SIT1(R3) PUT BYTE READ INTO APPROPRIATE BYTE IN SIT1
        INC  R3           POINT TO NEXT BYTE IN SIT1
        C    R0,R2        FINISHED ROW?
        JEQ  RC1          YES, SET UP FOR NEXT ROW
        DEC  R0           POINT TO NEXT COLUMN ON SCREEN
        JMP  RC           DO NEXT COLUMN, SAME ROW
RC1     CI   R2,>2E0      FINISHED SCREEN?
        JEQ  EXIT         YES, EXIT
        AI   R2,>20       SET R2=BEGINNING OF NEXT ROW
        JMP  ROW          DO NEXT ROW
EXIT    CLR  R0           -
        LI   R1,SIT1        | PUT SIT1 IN SIT AND
        LI   R2,>300        | BRANCH TO RETURN TO BASIC ROUTINE
        BLWP @VMBW          |
        B    @RETURN      -

SIT1    BSS  >300         FIRST BUFFER FOR DATA FROM SIT
SIT2    BSS  >300         SECOND BUFFER FOR DATA FROM SIT
VMBR    EQU  >202C        VIDEO MULTIPLE BYTE READ BLWP ADDRESS

PUTSI1  LWPI MYWS         SET WORKSPACE
        LI   R1,SIT1      GET READY TO READ INTO BUFFER SIT1
        JMP  PUTSI        USE COMMON ROUTINE
PUTSI2  LWPI MYWS         SET WORKSPACE
        LI   R1,SIT2      GET READY TO READ INTO BUFFER SIT2
PUTSI   CLR  R0           GET READY TO READ FROM FIRST SCREEN POSITION, 0
        LI   R2,>300      GET READY TO READ ENTIRE SCREEN
        BLWP @VMBR        READ SCREEN AND STORE IN BUFFER
        B    @RETURN      BRANCH TO RETURN TO BASIC ROUTINE
GETSI1  LWPI MYWS         SET WORKSPACE
        LI   R1,SIT1      GET READY TO WRITE FROM BUFFER SIT1
        JMP  GETSI        USE COMMON ROUTINE
GETSI2  LWPI MYWS         SET WORKSPACE
        LI   R1,SIT2      GET READY TO WRITE FROM BUFFER SIT2
```

```
GETSI   CLR   R0          GET READY TO WRITE TO FIRST SCREEN POSITION, 0
        LI    R2,>300     GET READY TO WRITE ENTIRE SCREEN
        BLWP  @VMBW       WRITE BUFFER TO SCREEN
        B     @RETURN     BRANCH TO RETURN TO BASIC ROUTINE

PDT1    BSS   >390        FIRST BUFFER FOR DATA FROM PDT
PDT2    BSS   >390        SECOND BUFFER FOR DATA FROM PDT
PUTPD1  LWPI  MYWS        SET WORKSPACE
        LI    R1,PDT1     GET READY TO READ INTO BUFFER PDT1
        JMP   PUTPD       USE COMMON ROUTINE
PUTPD2  LWPI  MYWS        SET WORKSPACE
        LI    R1,PDT2     GET READY TO READ INTO BUFFER PDT2
PUTPD   LI    R0,PDT      GET READY TO READ FROM PDT
        LI    R2,>390     GET READY TO READ ENTIRE PDT
        BLWP  @VMBR       READ PDT INTO BUFFER
        B     @RETURN     BRANCH TO RETURN TO BASIC ROUTINE
GETPD1  LWPI  MYWS        SET WORKSPACE
        LI    R1,PDT1     GET READY TO WRITE FROM BUFFER PDT1
        JMP   GETPD       USE COMMON ROUTINE
GETPD2  LWPI  MYWS        SET WORKSPACE
        LI    R1,PDT2     GET READY TO WRITE FROM BUFFER PDT2
GETPD   LI    R0,PDT      GET READY TO WRITE TO PDT
        LI    R2,>390     GET READY TO WRITE ENTIRE BUFFER TO PDT
        BLWP  @VMBW       WRITE BUFFER INTO PDT
        B     @RETURN     BRANCH TO RETURN TO BASIC ROUTINE

PABBUF  EQU   >1000        DATA BUFFER ADDRESS IN VDP
PAB     EQU   >0F80        PERIPHERAL ACCESS BUFFER ADDRESS IN VDP
PNTR    EQU   >8356        POINTER TO FIRST BYTE AFTER PAB
SV      BYTE  >06          FOR FIRST BYTE OF PDATA IF SAVE
LD      BYTE  >05          FOR FIRST BYTE OF PDATA IF LOAD
PDATA   DATA  >0600,PABBUF,>0000,>0000,>000F   PAB INFO (SEE P. 294 OF E/A MANUAL)
FNAME   TEXT  '                '               PAB INFO (FILENAME)
CONST   BYTE  >0F          CONSTANT TO RESTORE MAX. FILENAME LENGTH
ADDRES  DATA  0            POINTER TO ADDRESS OF SIT1 OR SIT2 OR PDT1 OR PDT2
SAVSI1  LWPI  MYWS         SET WORKSPACE
        LI    R0,SIT1      GET READY TO SAVE FROM SIT1
        JMP   SAVSI        USE COMMON ROUTINE
SAVSI2  LWPI  MYWS         SET WORKSPACE
        LI    R0,SIT2      GET READY TO SAVE FROM SIT2
SAVSI   LI    R1,>300      GET READY TO SAVE ENTIRE BUFFER
        JMP   SAV          USE COMMON ROUTINE
SAVPD1  LWPI  MYWS         SET WORKSPACE
        LI    R0,PDT1      GET READY TO SAVE FROM PDT1
        JMP   SAVPD        USE COMMON ROUTINE
SAVPD2  LWPI  MYWS         SET WORKSPACE
        LI    R0,PDT2      GET READY TO SAVE FROM PDT2
SAVPD   LI    R1,>390      GET READY TO SAVE ENTIRE BUFFER
SAV     BL    @STRETC      GET FILENAME FROM BASIC, ETC.
        MOVB  @SV,@PDATA   PUT SAVE DESIGNATOR IN PDATA
        BL    @SETPAB      ESTABLISH PAB IN VDP
        BL    @SETADD      ESTABLISH MEMORY AREA TO SAVE FROM
        BLWP  @VMBW        WRITE THE MEMORY AREA IN VDP
        BLWP  @DSRLNK      USE DEVICE SERVICE ROUTINE FOR DISK ACCESS
        DATA  8            DATA FOR DSRLNK
        B     @EXITD       BRANCH TO EXIT DISK ACCESS ROUTINES

LODSI1  LWPI  MYWS         SET WORKSPACE
        LI    R0,SIT1      GET READY TO LOAD TO SIT1
        JMP   LODSI        USE COMMON ROUTINE
```

```
LODSI2 LWPI MYWS          SET WORKSPACE
       LI   R0,SIT2       GET READY TO LOAD TO SIT2
LODSI  LI   R1,>300       GET READY TO LOAD ENTIRE BUFFER
       JMP  LOD           USE COMMON ROUTINE
LODPD1 LWPI MYWS          SET WORKSPACE
       LI   R0,PDT1       GET READY TO LOAD TO PDT1
       JMP  LODPD         USE COMMON ROUTINE
LODPD2 LWPI MYWS          SET WORKSPACE
       LI   R0,PDT2       GET READY TO LOAD TO PDT2
LODPD  LI   R1,>390       GET READY TO LOAD ENTIRE BUFFER
LOD    BL   @STRETC       GET FILENAME FROM BASIC, ETC.
       MOVB @LD,@PDATA    PUT LOAD DESIGNATOR IN PDATA
       BL   @SETPAB       ESTABLISH PAB IN VDP
       BLWP @DSRLNK       USE DEVICE SERVICE ROUTINE FOR DISK ACCESS
       DATA 8             DATA FOR DSRLNK
       BL   @SETADD       ESTABLISH MEMORY AREA TO LOAD INTO
       BLWP @VMBR         READ FROM VDP TO MEMORY
EXITD  MOVB @CONST,@PDATA+9  RESTORE MAX. LENGTH OF STRING FROM BASIC
       B    @RETURN          BRANCH TO RETURN TO BASIC ROUTINE

STRETC MOV  R0,@ADDRES    STORE ADDRESS OF DESIGNATED BUFFER FOR LATER USE
       MOV  R1,@PDATA+6   PUT # BYTES TO LATER ACCESS INTO PROPER WORD OF PDATA
       CLR  R0            -
       LI   R1,1            | GET FILENAME FROM BASIC
       LI   R2,PDATA+9      |
       BLWP @STRREF       -
       RT                 RETURN TO CALLER
SETPAB LI   R0,PAB        -
       LI   R1,PDATA        |
       LI   R2,25           |
       BLWP @VMBW           | SET-UP PAB IN VDP AND POINTER IN SCRATCH PAD
       LI   R6,PAB+9        |
       MOV  R6,@PNTR        |
       RT                 -
SETADD LI   R0,PABBUF     -
       MOV  @ADDRES,R1      | ESTABLISH AREA OF MEMORY TO ACCESS
       MOV  @PDATA+6,R2     |
       RT                 -

       COPY 'DSK1.GPLLNK'
       COPY 'DSK1.DSRLNK'
       END
```

```
> 100 CALL INIT :: CALL LOAD("
  DSK1.MIRROR/O")
> 110 CALL CHAR(140,RPT$("0",1
  2)&"FFFF",141,"FF",142,"80C0
  E0F0F8FCFEFF",143,"FEFCF8F0E
  0C08")
> 120 CALL CLEAR :: CALL HCHAR
  (12,10,140,2):: CALL HCHAR(1
  3,10,141,2):: CALL HCHAR(12,
  12,142):: CALL HCHAR(13,12,1
  43)
> 130 CALL HCHAR(1,1,49):: CAL
  L HCHAR(1,32,50):: CALL HCHA
  R(24,1,51):: CALL HCHAR(24,3
  2,52)
> 140 CALL LINK("PUTSI2")
> 150 CALL LINK("PUTPD2"):: GO
  SUB 410
> 160 A$=CHR$(143):: CALL LINK
  ("MIRROR"," ",A$)
> 170 CALL LINK("FLIP")
> 180 CALL LINK("PUTPD1"):: GO
  SUB 410 :: CALL CLEAR
> 190 CALL LINK("SAVSI1","DSK1
  .SIT1"):: CALL CLEAR :: CALL
   CHARSET :: A$="FINISHED" ::
   B$="SAVING" :: C$="SIT1" ::
   GOSUB 420
> 200 CALL LINK("SAVSI2","DSK1
  .SIT2"):: C$="SIT2" :: GOSUB
   420
> 210 CALL LINK("SAVPD1","DSK1
```

```
.PDT1"):: C$="PDT1" :: GOSUB
 420
> 220 CALL LINK("SAVPD2","DSK1
.1234567890"):: C$="12345678
90" :: GOSUB 420
> 230 CALL LINK("PUTSI1"):: CA
LL LINK("PUTSI2"):: CALL LIN
K("PUTPD1"):: CALL LINK("PUT
PD2")
> 240 CALL LINK("LODSI1","DSK1
.SIT1"):: B$="LOADING" :: C$
="SIT1" :: GOSUB 420
> 250 CALL LINK("LODSI2","DSK1
.SIT2"):: C$="SIT2" :: GOSUB
 420
> 260 CALL LINK("LODPD1","DSK1
.PDT1"):: C$="PDT1" :: GOSUB
 420
> 270 CALL LINK("LODPD2","DSK1
.1234567890"):: C$="12345678
90" :: GOSUB 420 :: GOSUB 41
0
> 280 CALL LINK("GETSI2")
> 290 CALL LINK("GETPD2")
> 300 CALL GCHAR(1,1,A):: CALL
 GCHAR(1,32,B):: CALL GCHAR(
24,1,C):: CALL GCHAR(24,32,D
):: GOSUB 410
> 310 CALL CHARPAT(50,A$):: CA
LL CHARPAT(143,B$)
> 320 CALL LINK("GETSI1")
> 330 CALL LINK("GETPD1")
> 340 CALL GCHAR(1,1,E):: CALL
 GCHAR(1,32,F):: CALL GCHAR(
24,1,G):: CALL GCHAR(24,32,H
)
> 350 CALL CHARPAT(50,C$):: CA
LL CHARPAT(143,D$):: E$=RPT$
("*",28):: GOSUB 410
> 360 CALL CLEAR :: CALL LINK(
"GETPD2"):: CALL CHARPAT(66,
Y$):: CALL CHARPAT(67,Z$)::
CALL CHAR(98,Y$):: CALL CHAR
(99,Z$)
> 370 DISPLAY AT(12,1):"ABCD":
"only the letters b and c":"
should mirror"
> 380 FOR I=1 TO 100 :: CALL L
INK("MIRROR","B","C"):: NEXT
 I :: GOSUB 410
> 390 CALL CLEAR :: PRINT A$:C
$:E$:B$:D$:E$:A:E:E$:B:F:E$:
C:G:E$:D:H:E$
> 400 END
> 410 FOR I=1 TO 500 :: NEXT I
 :: RETURN
> 420 DISPLAY AT(12,1):A$&" "&
B$&" "&C$ :: RETURN
```

Using Multiplan™'s ISERROR

by Richard M. Mitchell

While Multiplan™ has often been criticized for being somewhat slow in execution speed, it is a very impressive environment for the speed and ease of program development.

I prefer to think of Multiplan™ as a combination of a BASIC language, a text editor and a database. While it is generally referred to as a spreadsheet program, somehow that term loses the essence of Multiplan™'s capabilities and calls to mind a bookkeeper laboring over a huge report to arrive at a bottom line of dollars and cents. Though Multiplan™ can certainly be used handily for that purpose, it is by no means limited to that end.

Though Multiplan™ does offer a simplistic yet powerful approach to modeling solutions, there are some techniques that must be learned to make optimal use of Multiplan™ . For instance, Multiplan™ follows arithmetic rules and will not accept zero as a divisor, yielding a "#DIV/Ø!" error. Are we merely stuck when we have a set of data that would use zero as a divisor, even though we intend the answer to be expressed as zero in such a situation? No! Multiplan™ has an ISERROR function to trap out errors denoted as #NA, #VALUE!, #REF!, #DIV/O!, #NUM!, #NAME?, AND #NULL!.

To illustrate the use of ISERROR, we'll use the calculation of a baseball batting average. The batting average is calculated by dividing hits by times

at bat, usually expressed to 3 decimal places (understanding baseball is not really necessary). If a player has not yet batted, his average is usually said to be ".ØØØ", even though by arithmetic definition his average is undefined (it would sound rather silly if a television announcer stated that a player's average was "undefined", not to mention the ambiguity of such a statement!). Here is a Multiplan™ spreadsheet that calculates the batting average without use of ISERROR:

```
          1              2              3              4
1 "PLAYER"       "HITS"         "AT BATS"      "AVG."
2 "SMITH"        2              3              RC[-2]/RC[-1]
3 "JONES"        Ø              Ø              RC[-2]/RC[-1]
```

And, here's a report from the above sheet:

```
 PLAYER    HITS  AT BATS    AVG.
SMITH         2        3   0.667
JONES         Ø        Ø #DIV/Ø!
```

By adding an ISERROR as the first clause of an IF statement, we can trap out the error, as follows:

```
          1              2              3              4
1 "PLAYER"       "HITS"         "AT BATS"      "AVG."
2 "SMITH"        2              3              IF(ISERROR(RC[-
                                               2]/RC[-1]),Ø,RC
                                               [-2]/RC[-1])
3 "JONES"        Ø              Ø              IF(ISERROR(RC[-
                                               2]/RC[-1]),Ø,RC
                                               [-2]/RC[-1])
```

And, here's the report from the sheet that uses the ISERROR (much better):

```
 PLAYER    HITS  AT BATS    AVG.
SMITH         2        3   Ø.667
JONES         Ø        Ø   Ø.ØØØ
```

The IF function is equivalent to BASIC's IF-THEN-ELSE. To translate from "MPese" to English, if the batting average calculation produces an error, then the batting average is Ø, else the batting average equals the calculated value. By using relative cell references, such as RC[-2], we can simply Copy Down the formula into as many cells as we require -- in this case, for as many players as there are on a team or in a league, for example.

Perhaps at this point you may not yet recognize the power of Multiplan™. One advantage lies in the fact that data can be Sorted by any column, such as "AVG." or "HITS" or "AT BATS" or even in alphabetical order by the player's names without writing a sort routine. Sort is a Multiplan™ command! Also, there is little effort in laying out the screen or a printout -- simply move the cursor to wherever you wish to input! 99/4A users often remark that they'd like to have a word processor with a calculator function -- well, with Multiplan™ you have a text editor that will perform some of the functions of a word processor and it can be used as a calculator and much more! If you want to send fancy printer controls, you can print your spreadsheet to disk as a report and switch to TI-Writer to add your finishing touches, including such functions as incorporating the spreadsheet data into a form letter!

When using Multiplan™, it is highly recommended that Recalc be left off most of the time by selecting Options Recalc (No) or else the entire sheet

will be updated after every cell change. Don't worry about recalculating before saving the sheet to disk, as Multiplan™ will do that automatically!

Until next time, have fun with your spreadsheets!

---

256 BYTES OF SCRATCH PAD RAM - XB USE

>8300	XB TEMPORARY STORAGE AREA
	This area of Scratch Ram is used by X-Basic and Basic as a temporary holding area for the different routines.
>8300	temporary variable
>8302	temporary variable
>8304	temporary variable
>8306	temporary variable - Record Length on file access
>8308	temporary variable - Address of Sprite Atttribute List
>830A	temporary variable
>830C	temporary variable
>830E	temporary variable - increment value for Auto Num
>8310	temporary variable - used in CALL LINK parameter passing
>8312	temporary variable - used by CHAR type statements
>8314	temporary variable - copy of VDP reg 1 for some commands
>8316	temporary variable - DSR Link flag for some commands
>8318	XB PERMANENT STORAGE AREA
	This area of Scratch Ram is used for specific items by X-Basic
>8318	Used by LINK, LOAD & rtn control to Basic also String space bgn
>831A	Points to 1st free add in VDP RAM also String space end
>831C	Points to allocated str space - PAB Error - Temp string pointer
>831E	Start of current statement
>8320	Current Screen Address
>8322	Return error code from Assembly Language Code
>8324	VDP value stack base pointer
>8326	Return address from Assembly Language Code
>8328	NUD Table for Assembly Language Code.
>832A	Ending screen display pointer
>832C	Program text or token code pointer
>832E	Pointer to current line number in line number table
>8330	Start of Line number table pointer
>8332	End of Line number table pointer
>8334	Data pointer for read
>8336	Line number table pointer for read
>8338	Address of intrinsic Poly constants
>833A	Subprogram symbol table pointer
>833C	PAB address in VDP RAM (first link) PAB list
>833E	Symbol table pointer
>8340	VDP Ram free space pointer
>8342	Current char/token
>8344	Extended Basic Program RUN = 255 STOP = 0 (w/o 'READY')
>8345	Extended Basic System Flags
	Bit 0 1 = Auto-Num Bit 4 1 = Edit Mode
	1 1 = On Break Next 5 1 = On Warning Stop
	2 6 1 = On Warning Next
	3 1 = Trace 7
>8346	Crunch buffer destruction level
>8348	Last subprogram block on stack

256 BYTES OF SCRATCH PAD RAM Continued

>834A	FLOATING POINT and DSR usage, 36 bytes	
>834A	FAC (Floating point accumulator)	PAB I/0 OPCODE
>834B	for floating point routines	PAB FLAG/STATUS
>834C	this area holds a number in	PAB DATA BUFFER ADDRESS
>834E	radix 100 notation.	PAB LOGICAL REC LENGTH
>834F		PAB CHARACTER COUNT
>8350		PAB RECORD NUMBER
>8352		PAB SCREEN OFFSET
>8353		PAB OPTION LENGTH
>8354	FLOATING POINT ERROR CODE	PAB DEVICE LENGTH
>8356	SUBRTN POINTER / DSR's pnts to 1st char after PAB in VDP	
>8358		DSR
>835A		DSR
>835C	ARG (Floating point argument)	DSR
	and DSR usage	DSR
		DSR
>836C	FPERAD (float pnt err add in Grom ?)	DSR
>836D	Set to >08 for DSR call	DSR
>836E	INTERPRETER and FLOATING POINT GPL VALUE STACK POINTER	
>8370	HIGHEST AVAILABLE ADDRESS IN VDP RAM	
>8372	LSByte OF DATA STACK POINTER = A0 = (>83A0)	
>8373	LSByte OF SUBROUTINE STACK POINTER = 80 = (>8380)	
>8374	KEYBOARD NUMBER TO BE SCANNED Default =0	
>8375	ASCII CODE DETECTED by SCAN routine also SGN for float/point	
>8376	JOYSTICK Y-STATUS by SCAN routine also EXP for float/point	
>8377	JOYSTICK X-STATUS by SCAN routine	
>8378	RANDOM NUMBER GENERATOR RND's >0 ->63 (0-99)	
>8379	VDP INTERRUPT TIMER >0 ->FF (0-255)	
>837A	HIGHEST SPRITE # IN AUTO-MOTION >0 ->20 (0-32)	
>837B	COPY OF VDP STATUS REGISTER	
>837C	GPL STATUS BYTE (Set to 0 for a DSR CALL) (>20 =Key Press)	
>837D	CHARACTER BUFFER BYTE to VDP RAM screen table	
>837E	POINTS TO THE CURRENT ROW on the screen	
>837F	POINTS TO THE CURRENT COLUMN on the screen	
>8380	THE DEFAULT SUBROUTINE STACK (Used by GPL Routines)	
>8380	Reserved For Basics interpreter	
>8382	Reserved For Basics interpreter	
>8384	Reserved Highest Address in Expansion Memory	
>8386	Reserved Highest Free Address in Mem-Expansion	
>8388	Reserved For the Basic interpreter Sub stack base	
>8389	Reserved For the Basic interpreter Exp-Mem Flag	
>838A	RETURN ADDRESS STACK FOR GROM SUBROUTINES	
	(current Grom Address pushed to top of stack during Key Scan)	
>839E		
>83A0	THE DEFAULT DATA STACK (Used by GPL Routines)	
	this area holds various information according to the GROM	
	routine being executed.	
>83BF		

256 BYTES OF SCRATCH PAD RAM Continued

>83C0	INTERRUPT WORKSPACE REGISTERS	
>83C0	R0	RANDOM NUMBER SEED 2 Bytes >0-FF >0-FF
>83C2	R1	Bit 0 1 = disable ALL of the following
		1 1 = disable Auto Sprite Motion
		2 1 = disable Auto Sound Processing
		3 1 = disable The QUIT Key
		Bits 4-15 not used
>83C4	R2	ISR HOOK - Start address of User Interrupt Routine
>83C6	R3	Reserved for Keyboard state and debounce info
>83C8	R4	Reserved for Keyboard state and debounce info
>83CA	R5	Reserved for Keyboard state and debounce info
>83CC	R6	Pointer to Sound Table - also see >83FD
>83CE	R7	Number of Sound Bytes for Auto Sound Processing (0100)
>83D0	R8	Varies (>0000 for Cassette DSR Link)
>83D2	R9	Varies
>83D4	R10	CONTENTS OF VDP REGISTER 1 (used for key scan)
>83D6	R11	SCREEN TIME OUT COUNTER (blanks when incremented to 0000)
>83D8	R12	RETURN ADDRESS SAVED BY THE SCAN ROUTINE (Old R11)
>83DA	R13	Return WS for context switch (RTWP)
>83DC	R14	Return PC for context switch (RTWP)
>83DE	R15	Return ST for context switch (RTWP)

>83E0	GPL WORKSPACE REGISTERS (ALL Registers used by GPL interpreter)	
>83E0	R0	Varies NOTE: R0 - R7, R11 and R12
>83E2	R1	Varies are modified by Key Scan
>83E4	R2	Varies
>83E6	R3	Varies
>83E8	R4	Varies
>83EA	R5	Varies - Used by Interrupt Routine
>83EC	R6	Varies - Used by Interrupt Routine
>83EE	R7	Varies - Used by Interrupt Routine
>83F0	R8	Cleared on Return from Interrupt Routine
>83F2	R9	GPL Interpreter use
>83F4	R10	GPL Interpreter use
>83F6	R11	RETURN ADDRESS for BL instruction and User Interrupt
>83F8	R12	Varies - CRU Base Address for Key Scan and DSRs
>83FA	R13	GROM/GRAM READ DATA port (9800)
>83FC	R14	STATUS FLAGS
		Bits 0 - 7 Control the cursor blink speed & Auto sound processing. The value in this byte increments the counter at >8379
>83FD		Bit 0 4 1 = 16K Vdp Ram
		1 5
		2 1 = Cass Interrupt Timer 6 1 = Multi-Color mode
		3 1 = Cass Verify 7 Sound table location
		1 = VDP 0 = Grom/Gram
>83FE	R15	VDP WRITE ADDRESS port (8C02)

Oops!

In the November 1985 issue of *Super 99 Monthly*, the TI-Writer Dump program requires a modification. Add to the beginning of label L1Ø the following code:

```
L1Ø     MOVB @DEC3,R4
        CI   R4,>31ØØ
        JEQ  RET
```

Then, at the end of L1Ø, add a label at the RT, as follows:

```
RET     RT
```

For those with the Best of *Super 99 Monthly* disk, if your filename for the source code is "TIWDUMP-S", then you should make these modifications. If your filename is "TIWDUMP-S2", the changes already appear on both your source and object files. An error that was not in the publication exists in the source code on some of the disks. At line 264, under the label CONT2, a "?" appears instead of a ">". The corrected line should be:

```
LI   R0,>1F00
```

In the March 1984 issue of *The Smart Programmer*, in the Low Memory Expansion After CALL INIT map, the value at >2ØØ2, the FFAM, should be >24F4 instead of >24FA.

News

The State of Washington TI-99/4A Home Computer User Groups will sponsor The 1986 State of Washington TI-99/4A Convention September 27, at the Sea-Tac Holiday Inn, Seattle, Washington, with associated events scheduled Friday through Sunday. For more information, contact Barbara Wiederhold, 6 1/2 Boston St. #4, Seattle, Washington 981Ø9. Ms. Wiederhold can be reached at Queen Anne Computer Shoppe, phone (2Ø6) 283-Ø953. The phone line is operated as a BBS from 8pm to 8am PDT.

Some of you might be interested to know that I recently received some photographs of Heiner Martin's 8Ø-column video card, as well as photos of the monitor displays produced by the card. The card, which will be produced in Germany and may be available soon, was said by the photographer (a subscriber who was travelling in Europe) to be working properly. The card plugs into the I/O port on the right side of the console. For color displays, the card will require an RGB monitor (about twice as expensive as a regular color monitor, but the difference in quality between RGB and composite color is quite significant). For price and availability of the 8Ø-column card and other products from Germany, the U.S. distributor is T.A.P.E., Ltd., 1439 Solano Place, Ontario, CA 91764, U.S.A.

Version 3.3 of DM-1ØØØ, the popular disk manager Fairware program from the Ottawa User Group, is scheduled for release very soon. Among the updates for the release will be a 16-sector DS/DD option and support of up to 8 disk drives (an optimistic approach that goes beyond current hardware capabilities). The program is available from the Ottawa TI-99/4A U.G., Box 2144 Station D, Ottawa, Canada K1P 5W3.

P-Term, a widely-used terminal emulator program now at version 2.5, has been converted to the Fairware marketing concept. The program is now available on GEnie™.

Fast-Term users should note that the program was written for XMODEM D/F 128 file transfers to be compatible for use by other computers and therefore does not send a TI file header on such files. Barry Traver has discovered that protecting a file prior to uploads will trigger sending the TI file header. The program is currently at version 1.16. Later versions are scheduled to have a simplified option for sending TI file headers, as well as several other new features. Version 1.16 also has occasional problems with lengthy files (longer than >4Ø sectors), but a fix is now available on GEnie™.

Enhancements for XB and E/A!

Danny Michael's new enhancements package for Gram Kracker™ is now available from Millers Graphics! Danny, famous for his SCREEN DUMP and NEATLIST programs, has added many very impressive new features for TI Extended BASIC and Editor/Assembler (TI modules not included).

For Extended BASIC, here are some of the new features:

LIST will list with a designated output line length.
RES will resequence all or a part of a program.
TRACE can be output to a selected device and can be toggled on and off from within a program.
CALL QUITON will enable use of the QUIT key.
CALL QUITOFF will disable use of the QUIT key.
Screen and character colors have been modified.
Error messages will appear in upper and lower case.
Auto-load of file DSK1.LOAD can be by-passed with the press of any key.
New cursor control for program editing, INPUT's and ACCEPT's allows quick entries and editing.
COPY will copy blocks of one or more program lines, retaining the source lines.
DELETE will delete blocks of program lines.
MOVE will move blocks of program lines, deleting the source lines.
CALL EA will move directly to the Editor/Assembler.
CALL PEEKG will allow peeking GRAM or GROM addresses.
CALL POKEG will allow poking GRAM addresses.
CALL PEEKV will allow peeking VDP memory.
CALL POKEV will allow poking VDP memory.
All of the XBCALLS from the MILK disk will still be available (NEW, BYE, CLSALL, CLOCK, CLKOFF, CAT).
A new character set is placed in GRAM Ø.

For the Editor/Assembler, here are some of the enhancements:

Repeating keys.
Erase key.
Clear line to the right with <FCTN 4>.
The last filename accessed will always be retained (even after powering off).
Item 6 on the main E/A menu will be Extended BASIC, allowing moving directly to Extended BASIC.
Item 7 on the main E/A menu will be Format Ramdisk, which will format a MYARC Ramdisk with the equivalent of CALL PART and CALL EMDK.
Item 8 on the main E/A menu will be Catalog Disk.

Installing the Editor/Assembler and Extended BASIC simultaneously is optional.

The package will come complete with 22 pages of documentation, including details of the memory locations of the enhancements. Best of all, the price is a mere $1Ø plus $1.5Ø shipping and handling, available now from Millers Graphics.

New PROM For CorComp Controller

Millers Graphics will soon be releasing a new PROM for the CorComp Disk Controller Card. Here are the new features:

Gets rid of the CorComp title screen!
The CorComp Disk Manager can be accessed from BASIC or by holding down the space bar on power-up.
Several new CALL's will be available from BASIC or Extended BASIC, from a running program or from immediate mode, or from a Gram Kracker™ MSAVE'd BASIC program! Toolshed statements will also be available from MSAVE'd BASIC programs for the first time!
CALL LLR is a Link, Load and Run option that links to a Start name.
CALL ILR is an Initialize, Load and Run.
CALL LR is a Load and Run.
CALL RUN will run a Program Image file.
The four above-named CALL's are, of course, for linking to Assembly programs. The first three provide compatibility with the MYARC disk controller's routines of the same names.

A new feature will be available from virtually any environment (BASIC, Extended BASIC, Multiplan™, Editor/Assembler, TI-Writer, etc.). An asterisk (*) will denote a wildcard drive reference! Once a drive has been referenced once, the "*" will maintain that drive reference! Prior to referencing a drive, the default will be drive 1.

A price has not yet been set for the new PROM. Millers Graphics will be mailing brochures on their new products once a price has been set. If you are not on the MG mailing list, the address is Millers Graphics, 1475 W. Cypress Ave., San Dimas, CA 91773.

Just think what you'll be able to do from Extended BASIC with MG's new Gram Kracker™ Extended BASIC enhancements, MG's new CorComp Controller PROM, *Genial Traveler*'s XXB and our XB MIRROR all functioning at one time! Wow!

DM AID

by Richard M. Mitchell

Displaying graphics has been very popular among 99'ers recently, with many users turning to programs such as Display Master ($14.95 plus $1.50 U.S.A. shipping from Inscebot, P.O. Box 260, Arnold, MD 21012, U.S.A.). Display Master uses command files to pass instructions such as LOADPIC, PAUSE, DELAY, LOOP, etc. to the program. Pictures must be in TI-Artist format, so the pictures can be created directly through Inscebot's TI-Artist program or converted to TI-Artist format by MAX/RLE (see July '86 issue).

I was rather content in creating the Display Master command files through the E/A Editor until friends began sending several disks of Artist pictures at a time. Though I was very appreciative of the generosity of the senders, I soon wondered whether I would ever find time to view several DS/DD disks of pictures every week! There had to be a way to view the pictures automatically. My solution was to write a simple but useful program to create a Display Master command file. So, the Extended BASIC program listed below, which I call DM AID, will allow you to name your command file, checking to see that the filename selected is not already on the disk, and then create a command file to include all TI-Artist picture files on the disk (Artist picture filenames end with "_P"). Note that the command file should be written to the same disk drive as the drive you wish to read from via Display Master, as that is the drive the LOADPIC commands will reference.

```
> 100 DISPLAY AT(1,1)ERASE ALL
  :"DM AID":"THE SMART PROGRAM
  MER"
> 110 OPTION BASE 1
> 120 DIM N$(127)
> 130 DISPLAY AT(10,1):"NAME F
  OR DISPLAY MASTER     COMMAN
  D FILE:":"DSK1.CFILE"
> 140 ACCEPT AT(12,1)BEEP VALI
  DATE(UALPHA,DIGIT,".*")SIZE(
  -15):F$ :: DISPLAY AT(5,1):"
  ":""
> 150 OPEN #1:SEG$(F$,1,5),INP
  UT ,RELATIVE,INTERNAL
> 160 I=1
> 170 INPUT #1:A$,U,U,U
> 180 INPUT #1:N$(I),U,U,U
> 190 IF N$(I)=SEG$(F$,6,10)TH
  EN DISPLAY AT(5,1):"DUPLICAT
  E FILENAME,":"TRY AGAIN" ::
  CLOSE #1 :: GOTO 130
> 200 IF N$(I)<>"" THEN I=I+1
  :: IF I<128 THEN 180
> 210 CLOSE #1
> 220 OPEN #2:F$,DISPLAY ,VARI
  ABLE 80,OUTPUT
> 230 FOR J=1 TO I
> 240 IF POS(N$(J),"_P",2)<>0
  THEN PRINT #2:".LOADPIC "&CH
  R$(34)&SEG$(F$,1,5)&SEG$(N$(
  J),1,LEN(N$(J))-2)&CHR$(34)&
  ";":".DELAY 5;"
> 250 NEXT J
> 260 CLOSE #2 :: END
```

Display Master will generate a misleading error if a LOADPIC command fails, referencing the following command line, so carefully key line 240 of DM AID to avoid confusion.

BYTEMASTER ORDER FORM

The Smart Programmer

SP1	$18.00	U.S. AND CANADA FIRST CLASS
SP2	$15.00	U.S. THIRD CLASS (no back issues)
SP3	$20.00	FOREIGN SURFACE (no back issues)
SP4	$32.00	FOREIGN AIRMAIL
SP5A-C	$ 1.75	U.S. JUNE - AUGUST 1986, ea.
SP6A-C	$ 2.75	FOREIGN JUNE - AUGUST 1986, ea.

Super 99 Monthly

SM1	$18.00	Complete set of 18 back issues
SM2A-R	$ 1.00	Back issues - ea. (U.S. Third Class)
SM3A-R	$ 1.50	Back issues - ea. (Canada and U.S. First Class)
SM3A-R	$ 2.50	Back issues - ea. (Foreign Air Mail)
SM4	$12.00	Programs on disk (non-FORTH)
SM5	$15.00	Super 99 Handicapper (req. XB, 32K, Disk, Printer)

ITEM #	QTY	EACH	AMOUNT
_____	_____	_____	_____
_____	_____	_____	_____

NAME_____

ADDRESS_____

CITY_____

STATE_____

ZIP CODE_____

COUNTRY_____

Payments accepted by check or money order in U.S. funds, coded for processing through the U.S. Federal Reserve Banking System. No billings or credit sales. Dealer inquiries invited. Discounts available on quantity orders.

The Smart Programmer is published monthly by Bytemaster Computer Services, 171 Mustang Street, Sulphur, LA 70663.

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663-6724
U.S.A.

Bulk Rate
U.S. Postage
PAID
Sulphur, LA 70663
Permit No. 141

POSTMASTER: ADDRESS CORRECTION REQUESTED
RUSH -- TIME DATED MATERIAL