Vol. 7 No. 6
August, 1985

THE MSP 99 NEWSLETTER

JMP into Assembly Language with MMM

by Gordy Myers

The Mini Memory Module provides the beginning Assembly Language programmer an alternative to investing in the Memory Expansion and Disk System.

Using essentially the same program as the one presented by Ed Johnson, with a few considerations for the MMM and Line by Line Assembler, you can give your computer a new character set with real lower case letters with true decenders.

We will use the Line by Line Assembler that comes with the MMM. Note, however, there are other methods of entering Assembly programs with the MMM.

Before we begin; 'R E L A X'. This will be much easier than you might have imagined.

First load the assembler following the instructions on page 4 in the assembler manual. Use instruction 7.B to begin "assembling" a "NEW" program.

Press the space bar before typing each line below. Hit "Enter" at the end of each line. The four characters on the left are for reference. DO NOT type these! Also, the comments after the "*" do not need to be typed.

(Continued on Page 6)

SOFTWARE CONTEST UPDATE

The entries are all in and the judging is over. About 2 dozen programs have been submitted in this years contest and this month we get to find out who the actual winners are.

If any of you have not already heard, the MSP Officers have decided to upgrade the prizes in this years software contest. Top winners in each of the various categories will recieve a box of blank disks or tapes. (Winners choice.)

So, be sure to make it to this months meeting and see all the new additions to our growing Software Library.

The MSP 99 USERS GROUP meets each month for discussions and presentations that enable its members to be better informed about their computers. Users group members share and exchange information. Some members have a broad range of computer expertise, others are just beginning. We are not affiliated with or sponsored by any other group or company. Membership dues are $12 a year for a family, $10 for an individual, and $50 for a sponsor member. You're welcome to visit a meeting before you join. Call or write for more information.

USERS GROUP MEETINGS are held the third Tuesday of each month at Dunwoody Industrial Institute, 818 Wayzata Blvd., Minneapolis, MN 55403.

MSP 99 USERS GROUP
P.O. BOX 12351
ST. PAUL, MN 55112, U.S.A.

The MSP 99 NEWSLETTER is published eleven times per year, on a monthly basis, except during July, by the MSP 99 Users Group. Members are encouraged to contribute articles for publication. Opinions expressed are those of the writers and not necessarily those of the MSP 99 Users Group, its officers, editors, or members. Materials accepted by the editors for publication in the MSP 99 Newsletter, including software listings, are believed to be in the public domain. Newsletter articles may be reproduced by other users groups if appropriate credit is given to the author (if one is listed), and to the Minneapolis St. Paul 99 Users Group.

NEWSLETTER EDITORS:

Gary Gese

Articles intended for the next newsletter should be submitted NO LATER than the Users Group meeting on the month prior to publication. Articles submitted after this deadline are likely to appear in the following month's newsletter.

COMMITTEE VOLUNTEERS are sought for all of our committees. (Education, Equipment, Program, Publicity, Software, Newsletter) If you would like to join one of these committees or have an idea for a monthly program, please contact one of the officers.

COMMERCIAL ADVERTISEMENT RATES: Busines firms that wish to communicate with our members may do so by placing an advertisement in the newsletter. Rates are: Full Page (7-1/2 X 10-1/2) -- $40; Half Page (3-1/2 X 10-1/2 or 7-1/2 X 5) -- $30; Quarter Page (3-1/2 X 5) -- $22. Each ad must be camera-ready in one of the sizes indicated and paid in advance. Inserts (printed by the advertiser on 8-1/2 X 11 or 8 X 10) may be inserted in the newsletter at $20 per sheet. Contact the editors for information.

CHANGE OF ADDRESS: Before you move, please mail a change of address to the Users Group at the address listed in this issue. Please allow at least 1 month for processing.

MORE FREEWARE AVAILABLE

Last month's Freeware article was written by Danny Michael and yet failed to include any of his own Freeware offerings. These include an Assembly Language screen dump that works in XBASIC and uses an Epson/Gemini printer, and a utility entitled NEATLIST. It produces an orderly listing of your programs allowing you to set margins, and enter beginning and ending line numbers. It also gives you a list of all the variables used in the program.

Both these programs are available direct from Danny. Send an initialized disk, postage, and return mailer to:

Danny Michael
Route 9, Box 460
Florence, AL 35630

Ramsoft Enterprises is offering its Computer Craps game which originally sold for $14.95, as Freeware. It requires XBASIC Memory Expansion, and includes an instruction file.

Send an initialized disk + $2.00 to cover shipping and packaging to:

Ramsoft Enterprises
1501 E. Chapman Ave.
Suite 338
Fullerton, CA 92631

MICROpendium, a TI 99/4A magazine, is offering several disks of Freeware. These include, the updates for TI-Writer & Multiplan, Super Bugger, and an XBASIC loader for TI-FORTH.

The TI-Writer updates include new printer defaults and true lower case letters, while the Multiplan updates include an auto-repeating cursor. Both fit on one disk.

Super Bugger is an improved version of TI-Debugger and fits on one disk. while the XBASIC FORTH loader allows you to use TI-FORTH without the Editor/Assembler cartridge. This also requires one disk.

To order any of these, send the appropriate number of initialized disks to:

Freeware
c/o MICROpendium
P.O.Box 1343
Round Rock, TX 78680

MSP 99 Calendar of Events

AUGT 20: Software Contest Demos -- This is it. This month we take the time to see the winning entries of this years software contest. Prizes will be awarded at the meeting. Maybe we'll even see your entry here.

SEPT 17: Extended BASIC -- This month we take an indepth look at this powerful extension of our computers BASIC language; how is it different, and what it's capabilities and limitations are.

OCT 15: ANNUAL Module Swap & Auction -- This is it. The night many of you have been waiting for. Members are incouraged to bring down any Hardware or Software that they're not using any more for the Auction.

Subgroup Meetings

ASSEMBLY LANGUAGE--1st Tuesday of month, 7:00 p.m., Bryant Community Center, Bryant Ave and 31st St.

BUSINESS--Second Tuesday of month, 7-9 p.m., Call Bob DeMars (554-6219) for details.

EDUCATION--At monthly MSP 99 meetings.

YOUTH GROUP--At monthly MSP 99 meetings.

Committee Chairs

EQUIPMENT--George Madline (341-3780)

NEWSLETTER--Gary Gese

PROGRAM--Dick Dunbar (488-0153)

PUBLICITY--Dave Wunderlin (544-8266)

SOFTWARE--Steve Gonnella (533-8494)

YOUTH GROUP--Ed Johnson (690-3442)
Gordy Myers (377-6713)

ATTENTION -- ALL MSP 99 MEMBERS:

Effective January 1st, 1986, MSP 99 Memberships rates will be going up to $15.00 for either the Individual or Family memberships. Costs for the printing and mailing of the Groups Newsletter can be directly atributed to this rise in price, along with the purchase of 2 used color monitors & 2 consoles. We want to encourage everyone to renew their membership now before the increase.

For all of you that signed up for the Computer Shopper mini - subscription, and are still waiting for your first issue to arrive - relax. Your going to have to wait a little longer. It seems there was some sort of mix-up with our mailing date. However, things are on the mend and you should be recieving your first issue in the mail soon.

For all you Bulletin Board Users; there's a new BBS operating in the area called, the MOUND TIBBS. You can reach them at 472-3490. Settings are: Even Parity, 7 Data bits, 2 Stop bits. Give them a call and see what you think.

BEGINNING ASSEMBLY LANGUAGE DOESN'T HAVE TO BE SCARY!

by ED JOHNSON

It was the same story for me as it was for many others who purchased the Editor Assembler package for their TI computer. As I anxiously tore off the wrapper and began reading, I was dealt a severe blow on page 17, Chapter 1.1: "THIS MANUAL ASSUMES THAT YOU ALREADY KNOW A PROGRAMMING LANGUAGE," (o.k. so far - after all, I do get around BASIC and Extended BASIC pretty well!), "PREFERABLY AN ASSEMBLY LANGUAGE." Well, it didn't take long for me to get the message; this stuff was WAY over my head. I calmly closed the manual and put it away. Almost two years have passed since that fear-filled day and I finally found the courage to pick up the manual again about six months ago.

What helped get me started was the fact that some people were wise enough to publish some books geared toward people like me (they didn't assume a prior knowledge of Assembly Language programming). I also had the good fortune to find someone through a local bulletin board service (ComputerLine BBS 729-0408) who was willing to answer a lot of my questions (that person is Glenn Davis, who is a frequent contributor to this newsletter - Thanks Glenn!).

At this point I should also state that I have had no formal education in computer programming. Everything I have learned has been through self-education. My only motivation has been a desire to learn more about computers and programming (the more I learn, the more pleased I am that I chose a TI!). The point I'm trying to make is this: If I can do some useful assembly language programming, ANYONE CAN!

My intent with this article is to try to help others who are experiencing similar fears to better understand some of the "basics" that I have learned along the way. I am far from being an "expert", but I will do my best to try to explain those concepts and techniques that I do understand.

WHAT IS ASSEMBLY LANGUAGE?

Assembly language is a low-level language. It is lower than all other languages except for machine language, which is arguably the same thing. It is the language that the microprocessor itself understands. Languages like Pascal, BASIC, and Forth are examples of higher level languages. BASIC, for example is translated into machine code on a line-by-line basis.

The Editor/Assembler is one software package that allows you to write assembly language programs. This package is broken up into two parts: An Editor and an Assembler. The Editor allows you to create your program and the Assembler does the work of translating it into the machine code that the computer can use. The biggest advantage of running an assembly language program versus a BASIC program is that it runs considerably faster. The machine code generated by assembly language is loaded directly into the memory of the computer and executed - it doesn't need a "middle man" (the BASIC interpreter) to convert it like BASIC does. It is so much faster in fact, that many of the arcade games written in assembly language actually have delays to slow them down!

"THINKING" ASSEMBLY LANGUAGE

Probably the most difficult things for me to overcome in trying to write in assembly language is the fact that it is very difficult (if not impossible) to sit down with nothing more than an idea and start typing. Assembly language requires much more pre-planning. Most of the subprograms that are available in TI BASIC or TI Extended BASIC

are not there. You must design your own routines as the need arises (examples of this would be: CALL CLEAR, PRINT, DISPLAY AT, INPUT, ACCEPT AT, CALL CHAR, CALL SOUND, etc.).

As a beginner, I am interested in speeding up some of the routines that are slow in BASIC, and some of the slowest are those that deal with graphics; CALL CHAR, CALL GCHAR, CALL HCHAR, CALL VCHAR, and, DISPLAY AT.

What is really great about assembly language is that you don't have to write a program that "does it all". You can "LINK" subroutines that are written in assembly to your BASIC program. This is encouraging for me in that I don't have to learn everything at once - I can learn a little at a time and still see some success from what I've done. One of the first things I learned was how to re-define the whole character set faster than you can say "CALL CHAR".

USING THE EDITOR

The Editor is located on the disk labeled "Part A" and is loaded when you select "EDIT" or "LOAD" from the Editor's menu. Assembly language programs don't use line numbers like BASIC does. Labels (names) are used as one form of creating reference points. In X-BASIC you might have some commands like this:

```
100 X=0 ! BEGIN PROGRAM
...
... (other instructions)
...
250 GOTO 100 ! GO START PROGRAM AGAIN
```

In Assembly language you might use something like this to accomplish the same thing:

```
START  LI R0,0 ! BEGIN PROGRAM
...
... (other instructions)
...
   B    @START  ! GO START PRGM
```

Here is the general format that is used with the Editor in writing Assembly language programs:

```
[label]<sp><op-code><sp>
[<operand>][,<operand2>]<sp>
[comment]
```

The label must start in the first position of the line and can be up to six characters long. The label's first character must be alphabetic. Using a label is optional. The Op-code must be separated from the label by at least one space ("sp" above). The Op-code is the actual program instruction. The Operands are separated from the Op-code by at least one space. Depending on the instruction, you may need none, one, or two operands. If there are two, they must be separated by a comma and no spaces. Comments, like labels, are optional--but highly recommended since they take up no space in the finished program unlike BASIC REMs--and must be separated from the Operand(s) by at least one space.

Any time you wish to use a whole line for comments (as I do in the program below), just use an asterisk (*) as the first character in the line and it will be ignored during the assembly process.

The following program will instantly (well, almost instantly) re-define the entire Extended BASIC character set. The lower case characters will appear as "true" lower case (not as the small capitals they are upon power up). The rest of the characters have been modified slightly Also. Character codes above 127 are undefined in this program listing, but you may want to change the data statements for use in your own programs. One of the nicest features of programs like this is that they take up NO X-BASIC or BASIC memory space, but that is a topic for a forthcoming article.

After typing in this program, save it using the instructions in the Editor/Assembler manual (use "DSK1.CODES/SRC" for a file name). This is your "source" program. Saving it in VAR/80 format will conserve disk space.

After you have saved your program, load the Assembler to create your "object" program. The assembled object code is what X-BASIC will LOAD (bring into memory) and LINK (begin executing) to. When the Assembler is loaded, you will be prompted for some information. Use "DSK1.CODES/SRC" for the source file name and "DSK1.CODES/OBJ" for the object file name. "LIST FILE NAME?" is used if you want to make a printed listing of your program. Press <enter> if you don't want a listing or enter the device name for your printer.

The last prompt is for "OPTIONS". If you don't want a printed listing, enter just "R" (this tells the assembler that we are using register symbols R0 thru R15 as opposed to just the numbers 0 thru 15 - see the E/A manual, chapter 2 for more information). If you want a printed listing, enter "RLS" for the options and a listing will be produced on the device specified for the list file. There is a fourth option available, "C", which will produced a compressed object file. DO NOT CHOOSE THIS OPTION FOR PROGRAMS THAT ARE TO BE LOADED FROM EXTENDED BASIC. Extended BASIC does not recognize compressed files and will give you the "UNRECOGNIZED CHARACTER" error.

I have included a short X-BASIC program that will demonstrate how this program works. If you're like me you'll find that you have plenty of questions if you are new to Assembly language programming. My best advise is ASK THEM! I have never been accused of asking a dumb question. The only dumb question I know of is the one that's never asked. Anything you need to know can usually be found in the E/A manual if you do enough digging (it

(Continued on Page 11)

Minimem (Continued from Page 1)

```
7D00   AORG >7CD8   *Move to start
                     location of
                     our program.
7CD8   LI R0,>0400  *Location to
                     write data to
                     in VDP RAM
7CDC   LI R1,>7CEE  *Location in
                     CPU to find
                     beginning of
                     data
7CE0   LI R2,760    *Number of
                     BYTES to write
7CE4   BLWP >6028   *Branch and
                     link to
                     Multiple Byte
                     write routine
7CE8   CLR >837C    *Clear error bit
7CEC   B            *Return to
                     BASIC
7CEE   DATA >0000,>0000,. . . . .
```

This is, essentially, the program. The rest is the data that will contain the pattern identifiers of our new character set. Enter the data without Labels (SET1,SET2,etc.) through character 126. Remember to hit the space bar at the beginning of each line. After entering the data for character 126, enter the following lines.

```
     AORG >7FE8  *Move to location
                  we place program
                  name so the
                  BASIC CALL LINK
                  can find it.
7FE8   TEXT 'CODES' *Enter program
                     name CODES
                     with a trail-
                     ing space.
                     Enclose with
                     apostrophies.
     DATA >7CD8    *Location of
                     the start of
                     our program.
     END
```

Now here's the Assembly code:

```
*
*
* DSK1.CODES/SRC
*
*
* "CODES" - TO ACCESS THIS ROUTINE FROM X-BASIC USE THE FOLLOWING STATEMENTS:
*
*                                100 CALL INIT
* BY ED JOHNSON                  110 CALL LOAD("DSK1.CODES/OBJ")
*    12/15/84                    120 CALL LINK("CODES")
* (DATA COURTESY OF GLENN DAVIS)


       DEF  CODES   * THIS STATEMENT TELLS THE "LINK" STATEMENT IN BASIC WHERE
*                     TO FIND OUR PROGRAM IN CPU RAM

VMBW   EQU  >2024   * VDP MULTIPLE BYTE WRITE - THIS IS A ROUTINE THAT ALLOWS US
*                     TO WRITE DATA TO THE VDP RAM.  THIS NUMBER COMES FROM THE
*                     E/A MANUAL APPENDIX 24.4.8.
*
GPLWS  EQU  >83E0   * GRAPHICS PROGRAMMING LANGUAGE WORK SPACE - THIS IS THE
*                     ACTIVE WORK SPACE WHEN WE ENTER OUR ASSEMBLY LANGUAGE
*                     PROGRAM.  WE NEED TO KNOW IT'S LOCATION IN CPU RAM BECAUSE
*                     WE WILL BE USING OUR OWN WORK SPACE REGISTERS.  THIS WORK
*                     SPACE WILL HAVE TO BE RESTORED BEFORE WE CAN RETURN TO
*                     EXTENDED BASIC.

STATUS EQU  >837C   * THIS IS THE ADDRESS OF THE GPL (GRAPHICS PROGRAMMING
*                     LANGUAGE) STATUS BYTE.  THIS BYTE WILL BE CLEARED
*                     (RESET TO ZERO) BEFORE RETURNING TO EXTENDED BASIC
*                     SO THAT NO ERRORS OCCUR.

SAV11  BSS  2       * THIS STATEMENT IS RESERVING 2 BYTES OF MEMORY IN CPU RAM
*                     TO STORE THE CONTENTS OF REGISTER NUMBER ELEVEN OF THE
*                     GPLWS.  THIS REGISTER CONTAINS THE ADDRESS THAT WILL GET
*                     US BACK TO EXTENDED BASIC AND WILL HAVE TO BE RESTORED
*                     SO THE PROGRAM WILL SEND US BACK TO THE RIGHT PLACE.

OURWS  BSS  >20     * THIS STATEMENT IS RESERVING A BLOCK OF MEMORY IN CPU RAM
*                     FOR OUR OWN WORK SPACE REGISTERS.  THERE ARE SIXTEEN
*                     REGISTERS AVAILABLE FOR OUR USE (ALTHOUGH THIS PROGRAM
*                     WILL USE ONLY THREE OF THEM) AND EACH REGISTER REQUIRES
*                     TWO BYTES OF MEMORY (2=32 [decimal] = >20 [hex])
*
*
* THE FOLLOWING STATEMENTS CONTAIN THE DATA FOR RE-DEFINING CHARACTER CODES
* 32 - 143.  THE LABELS SET1,SET2,...,SET14 ARE THE BEGINNING OF EACH OF THE
* FOURTEEN CHARACTER SETS IN EXTENDED BASIC.  THE REMARKS WHICH FOLLOW THE DATA
* ARE THE CHARACTER AS IT LOOKS AT POWER-UP, IT'S ASCII NUMBER, THE DECIMAL
* ADDRESS IN VDP RAM THAT CONTAINS THE PATTERN DESCRIPTOR (THE SIXTEEN
* CHARACTER STRING THAT IS RETURNED WITH THE "CALL CHARPAT" COMMAND IN
* X-BASIC), AND FINALLY, THE SAME ADDRESS REPRESENTED IN HEXADECIMAL NOTATION.
* THE ASSEMBLER DISTINGUISHES BETWEEN DECIMAL (BASE TEN) AND HEXADECIMAL (BASE
* SIXTEEN) BY CHECKING FOR THE ">" SYMBOL.  ANY DATA (NUMBERS, ADDRESSES, TEXT,
* ETC.) THAT YOU WANT REPRESENTED IN HEX MUST HAVE THE ">" SYMBOL IN FRONT OF
* IT.
```

```
*
SET1    DATA >0000,>0000,>0000,>0000   *   *   32 * 1024 * >0400 *
        DATA >0010,>1010,>1000,>1000   * ! *   33 * 1032 * >0408 *
        DATA >0028,>2828,>0000,>0000   * " *   34 * 1040 * >0410 *
        DATA >0028,>7C28,>287C,>2800   * # *   35 * 1048 * >0418 *
        DATA >3854,>5038,>1454,>3800   * $ *   36 * 1056 * >0420 *
        DATA >6064,>0810,>204C,>0C00   * % *   37 * 1064 * >0428 *
        DATA >2050,>5020,>5448,>3400   * & *   38 * 1072 * >0430 *
        DATA >0008,>0810,>0000,>0000   * ' *   39 * 1080 * >0438 *
*
*
SET2    DATA >0810,>2020,>2010,>0800   * ( *   40 * 1088 * >0440 *
        DATA >2010,>0808,>0810,>2000   * ) *   41 *      *      *
        DATA >0028,>107C,>1028,>0000   * * *   42 *      *      *
        DATA >0010,>107C,>1010,>0000   * + *   43 *      *      *
        DATA >0000,>0000,>3010,>2000   * , *   44 *      *      *
        DATA >0000,>007C,>0000,>0000   * - *   45 *      *      *
        DATA >0000,>0000,>3030,>0000   * . *   46 *      *      *
        DATA >0004,>0810,>2040,>0000   * / *   47 *      *      *
*
*
SET3    DATA >0038,>4C54,>5464,>3800   * 0 *   48 * 1152 * >0480 *
        DATA >0010,>3010,>1010,>3800   * 1 *   49 *      *      *
        DATA >0038,>4408,>1030,>7C00   * 2 *   50 *      *      *
        DATA >007C,>0418,>0444,>3800   * 3 *   51 *      *      *
        DATA >0018,>2848,>7C08,>0800   * 4 *   52 *      *      *
        DATA >007C,>4078,>0444,>3800   * 5 *   53 *      *      *
        DATA >003C,>4078,>4444,>3800   * 6 *   54 *      *      *
        DATA >007C,>0408,>1020,>2000   * 7 *   55 *      *      *
*
*
SET4    DATA >0038,>4438,>4444,>3800   * 8 *   56 * 1216 * >04C0 *
        DATA >0038,>4444,>3C04,>7800   * 9 *   57 *      *      *
        DATA >0030,>3000,>3030,>0000   * : *   58 *      *      *
        DATA >0030,>3000,>3010,>2000   * ; *   59 *      *      *
        DATA >0810,>2040,>2010,>0800   * < *   60 *      *      *
        DATA >0000,>7C00,>7C00,>0000   * = *   61 *      *      *
        DATA >2010,>0804,>0810,>2000   * > *   62 *      *      *
        DATA >3844,>0408,>1000,>1000   * ? *   63 *      *      *
*
*
SET5    DATA >0038,>5C54,>5C40,>3800   * @ *   64 * 1280 * >0500 *
        DATA >0038,>447C,>4444,>4400   * A *   65 *      *      *
        DATA >0078,>2438,>2424,>7800   * B *   66 *      *      *
        DATA >0038,>4440,>4044,>3800   * C *   67 *      *      *
        DATA >0078,>2424,>2424,>7800   * D *   68 *      *      *
        DATA >007C,>4078,>4040,>7C00   * E *   69 *      *      *
        DATA >007C,>4078,>4040,>4000   * F *   70 *      *      *
        DATA >003C,>405C,>4444,>3800   * G *   71 *      *      *
*
*
SET6    DATA >0044,>447C,>4444,>4400   * H *   72 * 1344 * >0540 *
        DATA >0038,>1010,>1010,>3800   * I *   73 *      *      *
        DATA >0004,>0404,>0444,>3800   * J *   74 *      *      *
        DATA >0048,>5060,>5048,>4400   * K *   75 *      *      *
        DATA >0040,>4040,>4040,>7C00   * L *   76 *      *      *
        DATA >0044,>6C54,>5444,>4400   * M *   77 *      *      *
        DATA >0044,>6454,>4C44,>4400   * N *   78 *      *      *
        DATA >0038,>4444,>4444,>3800   * O *   79 *      *      *
```

```
*
SET7    DATA >0078,>4478,>4040,>4000  * P *  80 * 1408 * >0580 *
        DATA >0038,>4444,>5448,>3400  * Q *  81 *      *       *
        DATA >0078,>4478,>5048,>4400  * R *  82 *      *       *
        DATA >003C,>4038,>0404,>7800  * S *  83 *      *       *
        DATA >007C,>1010,>1010,>1000  * T *  84 *      *       *
        DATA >0044,>4444,>4444,>3800  * U *  85 *      *       *
        DATA >0044,>4428,>2810,>1000  * V *  86 *      *       *
        DATA >0044,>4454,>5454,>2800  * W *  87 *      *       *
*
*
SET8    DATA >0044,>2810,>2844,>4400  * X *  88 * 1472 * >05C0 *
        DATA >0044,>2810,>1010,>1000  * Y *  89 *      *       *
        DATA >007C,>0810,>2040,>7C00  * Z *  90 *      *       *
        DATA >0038,>2020,>2020,>3800  * [ *  91 *      *       *
        DATA >0000,>4020,>1008,>0400  * ¥ *  92 *      *       *
        DATA >0038,>0808,>0808,>3800  * ] *  93 *      *       *
        DATA >0000,>1028,>4400,>0000  *   *  94 *      *       *
        DATA >0000,>0000,>007C,>0000  * _ *  95 *      *       *
*
*
SET9    DATA >000C,>0804,>0000,>0000  * ` *  96 * 1536 * >0600 *
        DATA >0000,>3848,>4848,>3400  * a *  97 *      *       *
        DATA >0020,>2038,>2424,>7800  * b *  98 *      *       *
        DATA >0000,>3840,>4040,>3800  * c *  99 *      *       *
        DATA >0008,>0838,>4848,>3C00  * d * 100 *      *       *
        DATA >0000,>3844,>7C40,>3800  * e * 101 *      *       *
        DATA >0018,>2070,>2020,>2000  * f * 102 *      *       *
        DATA >0000,>3844,>443C,>0438  * g * 103 *      *       *
*
*
SET10   DATA >0060,>2038,>2424,>2400  * h * 104 * 1600 * >0640 *
        DATA >0010,>0030,>1010,>3800  * i * 105 *      *       *
        DATA >0008,>0008,>0808,>0830  * j * 106 *      *       *
        DATA >0020,>2428,>3028,>2400  * k * 107 *      *       *
        DATA >0030,>1010,>1010,>3C00  * l * 108 *      *       *
        DATA >0000,>A854,>5454,>5400  * m * 109 *      *       *
        DATA >0000,>5824,>2424,>2400  * n * 110 *      *       *
        DATA >0000,>3844,>4444,>3800  * o * 111 *      *       *
*
*
SET11   DATA >0000,>3824,>2438,>2020  * p * 112 * 1664 * >0680 *
        DATA >0000,>3848,>4838,>080C  * q * 113 *      *       *
        DATA >0000,>5824,>2020,>2000  * r * 114 *      *       *
        DATA >0000,>3C40,>3804,>7800  * s * 115 *      *       *
        DATA >0020,>7820,>2020,>1800  * t * 116 *      *       *
        DATA >0000,>4848,>4848,>3400  * u * 117 *      *       *
        DATA >0000,>4444,>2828,>1000  * v * 118 *      *       *
        DATA >0000,>6A2A,>2A2A,>1400  * w * 119 *      *       *
*
*
SET12   DATA >0000,>4428,>1028,>4400  * x * 120 * 1728 * >06C0 *
        DATA >0000,>2424,>241C,>0438  * y * 121 *      *       *
        DATA >0000,>7C08,>1020,>7C00  * z * 122 *      *       *
        DATA >0018,>2020,>4020,>2018  * { * 123 *      *       *
        DATA >0010,>1010,>0010,>1010  * ! * 124 *      *       *
        DATA >0030,>0808,>0408,>0830  * } * 125 *      *       *
        DATA >0000,>0020,>5408,>0000  * ~ * 126 *      *       *
        DATA >0000,>0000,>0000,>0000  *   * 127 *      *       *
```

```
*
SET13   DATA >0000,>0000,>0000,>0000   *    * 128 * 1792 * >0700 *
        DATA >0000,>0000,>0000,>0000   *    * 129 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 130 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 131 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 132 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 133 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 134 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 135 *      *       *
*
SET14   DATA >0000,>0000,>0000,>0000   *    * 136 * 1854 * >0740 *
        DATA >0000,>0000,>0000,>0000   *    * 137 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 138 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 139 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 140 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 141 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 142 *      *       *
        DATA >0000,>0000,>0000,>0000   *    * 143 *      *       *
*
* NOW WE GET TO THE "ENTRY" POINT OF OUR PROGRAM.  THE LABEL "CODES" IS THE
* SAME NAME WE DEFINED IN THE FIRST STATEMENT IN THE PROGRAM.  THIS IS WHERE
* THE "LINK" WILL BE MADE WITH OUR PROGRAM FROM X-BASIC.

CODES   MOV  R11,@SAV11    * SAVE THE LOCATION FOR RETURN TO X-BASIC IN THE BLOCK
*                            OF MEMORY WE RESERVED UNDER THE LABEL "SAV11"
*
        LWPI OURWS         * CHANGE REGISTERS TO OUR OWN
*                            "LWPI" MEANS "LOAD WORKSPACE POINTER IMMEDIATE"

*
* NOW WE HAVE TO SUPPLY SOME INFORMATION FOR "VMBW" SO IT CAN WRITE THE NEW
* CHARACTER CODES INTO THE PATTERN DESCRIPTOR TABLE OF VDP RAM.
* HERE'S THE INFORMATION THAT OUR FRIEND "VDP MULTIPLE BYTE WRITE" NEEDS:
*
* R0 (REGISTER 0)=> THE STARTING LOCATION IN VDP RAM TO WRITE THE NEW DATA
* R1 (REGISTER 1)=> THE LOCATION IN CPU RAM OF THE BEGINNING OF THE NEW DATA
* R2 (REGISTER 2)=> THE NUMBER OF BYTES THAT WE WANT "VMBW" TO WRITE
*
* THE COMMAND "LI" STANDS FOR "LOAD IMMEDIATE" AND SIMPLY PLACES THE
* INFORMATION NEEDED DIRECTLY INTO THE REGISTER INDICATED.  THE FORMAT IS:
*
        LI   R0,>0400      * LOC IN VDP RAM (THIS IS IN HEX NOTATION)
        LI   R1,SET1       * LOC IN CPU RAM (THE LABEL "SET1" IS WHERE THE DATA
*                            STARTS)
        LI   R2,112*8      * # BYTES TO WRITE - EACH CHARACTER REQUIRES 8 BYTES
*                            TO DEFINE IT.  WE ARE RE-DEFINING 112 CHARACTERS SO
*                            WE WILL HAVE TO WRITE 112*8 BYTES (896).  THE
*                            ASSEMBLER WILL DO MATH (+,-,/,*), BUT ONLY LEFT TO
*                            RIGHT, NO PARENTHESES ALLOWED.  SYMBOLS MAY ALSO BE
*                            USED, SO THE NUMBERS DO NOT NEED TO BE KNOWN BY
*                            THE PROGRAMMER.
*
* REGISTERS 0-2 NOW HAVE ALL THE NECESSARY INFO.  THE FOLLOWING COMMAND "BLWP"
* MEANS "BRANCH AND LOAD WORKSPACE POINTER".  THIS COMMAND IS SIMILAR TO
* "GOSUB" IN BASIC; THAT IS, IT BRANCHES TO ANOTHER LOCATION TO EXECUTE A
* SERIES OF COMMANDS AND RETURNS WHEN FINISHED.  IN THIS PROGRAM WE BRANCH TO
* "VMBW" TO COPY THE NEW CHARACTER CODES INTO THE PROPER PLACE IN VDP RAM.
*
        BLWP @VMBW         * WRITE THE PATTERN CODES
```

```
*
* WE HAVE NOW RE-DEFINED ASCII CHARACTERS 32-143 AND CAN RETURN TO X-BASIC.
* FIRST WE CLEAR A REGISTER (CLR), THE CONTENTS OF WHICH WE WRITE TO THE
* GPL STATUS REGISTER BY MOVING A BYTE THERE (MOVB).  THIS IS TO ENSURE
* THAT OUR PROGRAM DOESN'T CREATE ANY PSEUDO-ERRORS ON RETURN TO X-BASIC.
* NEXT THE GPL WORKSPACE REGISTERS ARE RE-LOADED.  LWPI=LOAD WORKSPACE
* POINTER IMMEDIATE.  OUR SAVED RETURN ADDRESS IS THEN MOVED BACK TO
* GPL R11 SO WE CAN RETURN TO X-BASIC.  NEXT THE PSEUDO-INSTRUCTION "RT"
* IS EXECUTED TO RETURN TO X-BASIC.
*
       CLR  R0              * CLEAR A REGISTER
       MOVB R0,@STATUS      * MOVE A ZERO BYTE TO GPL STATUS REGISTER
       LWPI GPLWS           * THE MAIN ROUTINE IS FINISHED SO GO BACK TO X-BASIC
       MOV  @SAV11,R11      * RESTORE RETURN ADDRESS
       RT                   * SAME AS B  *R11 (SEE E/A MANUAL FOR MORE INFO)
       END
```

ASSEMBLY: (CONTINUED FROM PAGE 6)

is NOT my favorite reading material!). Good luck and happy programming!

Here's the X-BASIC demo program:

```
100 CALL CLEAR
110 CALL INIT
120 CALL LOAD("DSK1.CODES/OBJ")
130 FOR X=32 TO 143 :: PRINT
CHR$(X);:: NEXT X :: FOR X=1 TO 12
:: PRINT :: NEXT X
140 DISPLAY AT(20,1):"PRESS ANY
KEY TO RE-DEFINE":"the
characters.....":"(make sure you
don't blink!)"
150 CALL KEY(0,K,S):: IF S=0 THEN
150
160 ! THE DURATION OF THE FIRST
CALL SOUND SHOWS THE SPEED OF THE
ROUTINE (.035 SECONDS)
170 ! A DURATION LOWER THAN 35
WILL ALLOW YOU TO HEAR A DISTINCT
BREAK BETWEEN THE TONES
180 CALL SOUND(-35,880,0)
190 CALL LINK("CODES")
200 CALL SOUND(-1,880,0)
210 ! TO USE THIS ROUTINE WITH
YOUR OWN PROGRAMS, USE LINES:
110,120 & 190.
220 GOTO 220
```

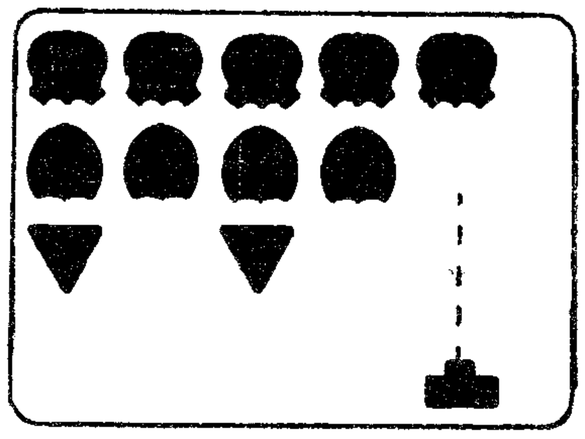
MINIMEM: (CONTINUED FROM PAGE 6)

You should have 0000 UNRESOLVED REFERENCES. Press enter twice then QUIT. Save the program following the directions on the back of the assembler manual. You now have in MMM an Assembly Language routine that can be "CALL"ed from your BASIC programs.

Let's see the new character set and the speed of Assembly Language. With the MMM in the cartridge slot enter this BASIC demo program.

```
100 CALL CLEAR
104 REM USE LINE 105 ONLY IF YOU
ARE USING THE EDITOR/ASSEMBLER
SOURCE FILE WITH EXTENDED BASIC
105 CALL LOAD("DSK1.CODES/OBJ")
110 FOR X=32 TO 126
120 PRINT CHR$(X);
130 NEXT X
140 PRINT
150 PRINT "PRESS ANY KEY TO
REDEFINE":"the characters . . .":
"(make sure you don't blink!)"
160 CALL KEY(0,K,S)
170 IF K=0 THEN 160
180 CALL LINK("CODES")
190 INPUT A$
200 GOTO 190
```

The GAME ROOM



FASTER BASIC

by Brian Beary

Reprinted from the Spirit of 99 Newsletter of the Central Ohio Ninety-Niners Users Group, Oct 1984.

This month I will provide some of the techniques to add to the speed and efficiency of game programs. Note that some of the tricks described below are not just limited to use in games, but might also help in drawing programs, Icon-driven software (like the Macintosh's), and many others.

One of the biggest time-wasters in game programs is the user input routine, whether from the keyboard or joysticks. Most of the time, Extended BASIC programmers end up trading off between less frequent input, allowing less control by the player but more time to perform other routines, and more frequent input, which gives the player better control of his spaceship, car, or whatever but limits the computer's response, often making the game easy.

Neither of these methods is particularly satisfying, and both, if not handled well, can produce dull games. In my opinion, the best solution is to maximize the speed of the routines you use, thus giving you more flexibility and ultimately, a better game.

On to the methods. First of all, the joystick routine will probably be the most executed line in most action games. Therefore, it makes good sense to make that routine as efficient as possible. Take for instance, this code:

```
500 CALL JOYST(1,X,Y):: Y=-Y ::
 X=X/4 :: Y=Y/4 :: CALL
 MOTION(#1,Y,X):: GOTO 500
```

While only taking up one line, this is wasteful. First of all, unless you need to store the joystick input for later use, you may make all calculations inside the CALL MOTION. So a better answer would be as follows:

```
500 CALL JOYST(1,X,Y):: CALL
MOTION(#1,-SGN(Y),SGN(X)):: GOTO
500
```

Note that the SGN produces the same effect, in this case, as dividing by 4 would. SGN returns a 1 if positive, 0 if the number is 0, and -1 if the number is negative.

If you need to use a certain formula many times in a program, you can save memory by using a DEF statement. TI BASIC and EXTENDED BASIC have a very powerful version of the DEF statement. They can be used for mathematical functions:

```
10 DEF CUBE(X)=X^3
```

Then when CUBE(3), for example, is referred to, it will return a 27. The X, if it is in parenthesis, wil have nothing to do with a variable named X elsewhere in your program.

In that lies the power of the DEF statement. You can use it to update a variable and print it in one command:

```
10 DEF XYZ=C*A^B
20 FOR C=1 TO 10 :: FOR A=1 TO 5
STEP >1
30 B=RND :: PRINT XYZ :: NEXT A
:: NEXT C
```

So after being set up, XYZ can take care of itself through the rest of the program. Another use could be quick solving of mathematical problems:

```
10 DEF ANSWER=A/(C+B)^A
20 INPUT A,B,C :: PRINT ANSWER ::
GOTO 20
```

But enough of these DEF statements. Keyboard input, while allowing more variety and number of controls, can be very cumbersome and difficult to handle. For instance:

```
200 CALL KEY(0,K,S):: IF S=0 THEN
1000
210 IF K=69 THEN CALL
MOTION(#1,-10,0):: GOTO 1000
220 IF K=83 THEN CALL
MOTION(#1,0,-10):: GOTO 1000
230 IF K=68 THEN CALL
MOTION(#1,10,0):: GOTO 1000
240 IF K=88 THEN CALL
MOTION(#1,0,10):: GOTO 1000
250 GOTO 1000
```

This method, while providing four directions, uses up a lot of time, and does not provide for diaginals. A better way:

```
200 CALL KEY(0,K,S):: IF S=0 THEN
1000
210 X=(10*(K=87 OR K=69 OR
K=82))+((-10)*(K=90 OR K=88 OR
K=67))
220 Y=(10*(K=87 OR K=83 OR
K=90))+((-10)*(K=82 OR K=68 OR
K=67)):: CALL MOTION(#1,X,Y)::
GOTO 1000
```

This method allows for diagonals, and is more efficient than the first version.

If you need more accuracy then automatic sprite motion can give, and are willing to trade off execution speed for more accuracy, you should try updating the sprites yourself using CALL LOCATE. For instance:

```
100 CALL CLEAR :: CALL
MAGNIFY(4):: CALL
SPRITE(#1,42,2,96,96):: V,H=96
200 CALL JOYST(1,X,Y)::
V=V+(SGN(-Y)*(V>1 AND V<180))::
H=H+(SGN(-Y)*(H>17 AND H<224))
210 CALL LOCATE(#1,V,H):: GOTO 200
```

Thus, (SGN(-Y)*(V>1 AND V<180)) checks to see what direction is desired, and then doublechecks to make sure that the move won't exceed the screen boundaries.

(CONTINUED ON PAGE 16)

TIPS 'N' THOUGHTS

by Tom Fairbairn

Your Terminal Emulator II module has the capability of saving information to diskette. The TE II module saves displayed data files screen by screen, and in DIS/VAR 80 format. This means that data written by TE II can be edited and printed using TI-WRITER.

Documents created using TI-WRITER can be sent to other systems using the TE II module. Also, documents created on an another TI system can be received on your system and printed by TI-WRITER just as if you had created them yourself. This makes it possible to bring in documents from several sources for, say, a newsletter or report, and merge them into a single printed document, all without ever mailing or touching a single piece of paper.

To do this, you use the ordinary disk-to-disk transmit and receive functions of TE II. Any edit commands, formatter commands, and margin/tabset rulers are kept intact when the file is transmitted. If the file you are transmitting happens to include MULTIPLAN printable files as a part of the TI-WRITER material, they will also be sent without problems.

As a matter of fact, this column is sent to the newsletter editors in just exactly this manner. I keep a proof copy of each month's entry and a copy on disk for archives, but no paper changes hands.

You may also quote directly from any BBS or news service that you can "talk" to with the conversational mode of TE II. You may save screen information on the disk, and this information is saved in the TI-WRITER DIS/VAR 80 format. A file created in this way may be edited using the TI-WRITER Editor, or be printed via the TI-WRITER Formatter.

The screen-to-disk output is controlled in the same way that you would print directly from the TE II screens using the CTRL/2 (Output) method. An interesting side effect of this capability is that using the TI-WRITER in this way saves having to write a seperate display/print routine as is suggested by the TE II manual. You can display the file on the screen or print it in Edit mode; you can print it with formatting in the Format mode.

You can NOT generate files with TI-WRITER to be sent to a bulletin board, however. Terminal Emulator II can not read files from a disk to be sent out as a direct text entry or input to an outside system.

In general, files being read from other than another TI system would be copied in the "conversational" mode of TE II, in the same way you would copy from a bulletin board. There are a couple of ways you may do this.

First, you may use the WRAP mode of TE II, with a screen width of 36, 38, or 40 characters. In this case, the received data wraps within the TI screen and you can receive up to 6 screens of wrapped data before the buffer fills up. The second method is to disable the WRAP feature. In this case TE II forms 80-column lines and you can "window" across the data much in the same way as with TI-WRITER. In the latter method, you may save up to three screens before the buffer is full. When the buffer fills, you will hear a beep tone. At that time, the incoming data begins to overwrite what was received previously.

To save a conversational file for use with TI-WRITER, take the following steps.

1) Initiate the transfer in the normal manner for the system you are receiving data from. Save up to 6 screens full in 40 column mode, or 3 screens full in 80 colm.

2) Stop the incoming data stream using CTRL/S (most other systems will accommodate this, but for a given system you should check).

3) Display the first screen of data received from the other system. Select the TE II OUTPUT function. At the prompt, enter your drive number and filename in the format DSKn.filename. TE II now writes the screen to the selected disk.

4) Display the next screen. Press OUTPUT. At the prompt "OUTPUT TO PREVIOUS DEVICE" enter 1 for the "yes". The next screen is now written to the disk. Repeat this process for each screen of data you have in the buffer.

5) Restart the data from the other system using CTRL/Q. Allow the next 3 or 6 screens of data to come into the buffer. Again stop the transfer with a CTRL/S.

6) Repeat steps 4 and 5 as many times as necessary to transfer and record the entire file of information you wish to save. When you have moved the entire file, continue on with the next step.

7) After you have written the last data, you need to close the file on the disk. To do this, again press OUTPUT, but at the prompt "OUTPUT TO PREVIOUS DEVICE" answer 2 for "no". This will close the file on the disk, leaving it in DIS/VAR 80 format. You may then return to the conversational mode of TE II by pressing the CANCEL key.

Files transmitted between TI systems using the FILE TRANSFER function retain all physical characteristics of the original file. Program files will thus still be program files when you receive them. DIS/VAR 80 files will also remainas such, so sending MULTI-PLAN and TI-WRITER files using TE II is very simple and the directions in the TE II manual are quite adequate.

In any event, be certain that you direct the output to a drive

containing a formatted diskette that is not write protected. If you address it to the wrong drive, or there is a problem with the selected diskette, the TE II program may just simply lock up.

By the way, source files with the protect bits set will be copied with the protect bits still set, so don't hope to use TE II for stripping off the protection.

I have used TE II to work with the CONTROL DATA PLATO (R) system, in what the system calls "Scrolling ASCII" mode. There are no graphics used in this mode (PLATO strips any graphics out of existing files in this mode). But I can use the screen printing and screen saving capabilities of TE II to copy information into my own files for later use or to get plain-Jane copies of news items and my efforts from work. The use of TE II for PLATO is the subject of another edition of this column.

TE II can also be used to copy a file or files from one TI to another locally. You have to build an RS-232 cable to run from one to the other machine, and you have to "twist" pins 2 and 3, 4 and 5, and 6 and 20 to make it work. You can then treat the two machines as if they were talking via the phone lines and modems for transferring data. It is a shame the TE II is so limited on speeds for this type of thing, since you should be able to run at the maximum I/O rate of the TI. TE II will not allow you to do so because of program limitations.

FORTH Information

The LA 99ers User's Group is offering its TI-FORTH Notes to TI users. There are 3 volumes that include screens and tips on how to get more out of TI-FORTH. Volumes 1 and 2 are $1.75 each and volume 3 is $2.50. All are postage paid. For more information, contact the group at:

P.O.Box 3547
Gardenia, CA 90247

MANAGING YOUR DISKS

AN MSP 99 SOFTWARE REVIEW

You may or may not be aware that our Software Library has aquired a new addition. This is aside from the many contest entries we'll be reviewing for you next month.

It is called DM1000 and it is one of the most useful sfotware packages to come along since Sprite Builder. It is a disk based Disk Manager that operates very similar to the Cor Comp Disk Manager. Two versions of the program are available; one for use with Extended BASIC, and another that is accessed through TI-Writer using option 3 on the main menu. Both versions are identical in operation.

Some of the features this program gives you are; the ability to copy, delete, rename, or write protect any program or file directly on the catalog listing. You can rename a disk, initialize a disk or a whole bow of disks, or copy a disk using either the full sector copier or the "Bit Map" mode which will copy only those sectors that contain data.

You can copy protect an entire disk with just the touch of a button, or just as easily remove the protection. There is even a very useful utility that will allow you to recover a deleted file as long as nothing new has been written to the disk since the file was deleted.

The program is another offering from the world of FREEWARE, this one is written by Bruce Caron, and it is the quality of program you would expect to pay big bucks for anywhere else. If you would like a copy of this very useful program, contact Software Chairman Steve Gonnella either by mail or at the monthly meetings. Be sure to specify whether you wish the XBASIC or TI-Writer version of the program. Once you try it, you won't want to be without it.

GAMEROOM: (CONTINUED FROM PAGE 13)

However, once those limits are reached, no further motion in that direction or the opposite is possible. Puzzle of the month: Rewrite the formula so that you may move back after you have hit one of the limits, without using an IF THEN construct.

The system described allows you to maintain complete accuracy as to the sprite's location.

Until next month...

WANT ADS

FOR SALE -- 300 Baud Accoustical
Modem with TI cable ... $40.00
TI Color Monitor ... $175.00
Cass Recorder & cable ... $25.00

Contact Ed Johnson 690-3442

FOR SALE -- 1 used Projection TV system. As seen at the Monthly MSP 99 meetings. Needs a reverse video monitor to replace the one that was stolen from the group. Open for offers. Contact Dick at 488-0153 or Jeff at 222-7099.

```
**************************
*ADIPLEOFHCISTFKSARKV*
*QDLAZKJCMZNXKSOLQFDK*
*SKEZNVBUTHGWPCSJBEWJ*
*ONJODJMYEMBPARSECNCR*
*VJHRFBUAWZKDNCIEUMEN*
*SKEENVNSJWOCKLSHVKER*
*AXYZWZCERUTNEVDAACVB*
*NJUTIXHSSJKWQLKENBZI*
*ILASWXMUCFGNSLRHDIOP*
*ISQNCJAPKUPSIBMPOSCT*
*HNTYMINELOWEWNACBAYZ*
*NOVAITERFGWABCNXRJKE*
*QOPARIMFVHJURZKWLQRY*
*BABPDTALCNVKEIALSOIU*
*EMLGEERYXIOEPRTHYNAB*
*ROARIRRENVITSKMCLIPZ*
*TNSCVJNSKALSOCMEUTVS*
*KDTMALPINERSURNVYWOV*
*QCOXEMITREGRUBCVYPRT*
*CMSUDMOONMINECISERUP*
**************************
```

WORD LIST

ADVENTURE	MOONMINE
ALPINER	MUNCHMAN
BLASTO	PARSEC
BURGERTIME	QBERT
CARWARS	STARTREK
HOPPER	SUPERFLY
INVADERS	ZEROZAP
JAWBREAKER	

MSP 99 USERS GROUP
P. O. BOX 12351
ST. PAUL, MINNESOTA 55112

Address Correction Requested

Bulk Rate
U.S. Postage
PAID
Permit #1285
Minneapolis, MN

TIME SENSITIVE MATERIAL
POSTMASTER - PLEASE DELIVER PROMPTLY