MICROPOLIS USERS GROUP

MUG Newsletter #30 - January 1983

****

SORTING BASIC ARRAYS (PART 2)

by Burks A. Smith of DATASMITH
Box 8036, Shawnee Mission KS 66208

Last month we discussed the Shell sort method and developed a simple program in BASIC that would sort a string array. We defined the sort data as fixed-length strings (records) with each record being composed of one or more fixed-length fields. The program sorts the records according to a a set of keys, which designate the starting position and length of the key data within the record string. The first key is most important, with additional keys being used if the first keys are equal.

This month we present an 8080 or Z-80 assembly language program which uses the same technique as the BASIC program, although they have slightly different logic. The assembly language program will sort a one-dimensional string array A$(x) on several keys in only a few seconds and can be called from BASIC. Actual times will vary according to data, but there is approximately an 80 to 90% reduction in the time BASIC would require using the same program.

Micropolis Basic allows the use of assembly language subroutines with the use of DEF FAx, which defines the routine's execution address. Basic will pass up to three parameters to the routine and expects to get data back, since this linkage works like a function in Basic. In our application, we have more than three parameters to pass and don't care to get a value back, since this program is supposed to sort an array in memory. However, we are bound to language convention so Basic passes nothing and the program returns a string value "A" if the operation was successful, and "E" if it was not. This particular version only returns "E" if the array A$(x) is anything but a one-dimensional array. Since the program returns a string you use it by setting it equal to a string. Example: Q$=FAS, where FAS has been defined as the sort function. You can ignore Q$, but you have to use it or some other string.

To pass the sort parameters and keys to the function it is convenient to use POKE statements. One 16-bit value, the number of elements in the array to be sorted, and one or more three byte keys designating start, length and sorting sequence. The POKE statement is especially useful because we want to convert Basic's binary coded decimal number format to 8-bit binary values that the processor can use directly. In the case of the 16-bit value, the Basic program calculates and POKEs two bytes in the proper Intel low-high format expected by the microprocessor. To keep everything together, the first three bytes the program are a jump instruction to the main program, with the parameter data following. The size of array parameter is stored at FAS+3, and the keys start at FAS+5. The program has provided for storage of up to 10 keys, with a START value of zero indicating the LAST KEY + 1.

The sorting program is composed of several subroutines representing different logical levels of the program. The main routine, starting at the label SORTLOOP makes processing passes through the array, calling the SWAP routine to compare data and swap if necessary. The routine makes as many passes as necessary through the array with a certain index between the two data pointers until no swaps need to be made to put the data in order. It then divides its index by two and repeats the process until the index value has been reduced to zero. Control is then passed back to Basic.

Most of the work is done by the SWAP routine, whose job it is to compare two strings. When this subroutine is called, it expects the array indexes of the data to be compared in the DE and HL registers. It calculates the actual memory address of the data to be compared and calls the KEYCOMP subroutine. Depending on the result of the comparison and whether or not an ascending or decending sort has been specified, the data is exchanged and the routine returns. The calculation of the address of data in memory is accomplished by knowing the address of the start of the array A$, the maximum length of each string, and the data's index in the array. In Micropolis Basic version 4 the 16-bit address stored at location 33B9H points to the beginning of array A$ and the maximum length of each string is in the array's header information. In effect, the index is multiplied by the length of each string and this value is added to the array's starting address. Since the processor can't actually multiply, the length of a string is successively added to the array address.

The balance of the program consists of additional subroutines that do the the actual comparing, swapping, and housekeeping functions of the routines at a logically higher level. The source code is commented, so if you are familiar with assembly language you shouldn't have any trouble figuring out how the program works. If you don't know assembly language, be very careful when implementing the program and don't leave anything out. While this program is designed to order blocks of memory, it can just as easily create chaos if it gets away from you. For example, one of the parameters POKEd is the number of elements to sort in the array. The program doesn't check to see if this exceeds the array's dimensioned size (although it could), so it could be made to "sort" outside of the array and scramble other data or system parameters like Basic's subroutine stack. It also doesn't check to see if a key's starting position is within the string. Garbage In, Garbage Out, if you are lucky. If you aren't lucky it will crash your system.

The program uses less than 512 bytes of code and should be placed as high as possible in memory, above Basic. The program illustrated assumes a 48K system, but the ORG can be changed to suit any system. Vector Graphic owners with Qume printers should avoid using the block of high memory above the screen RAM, since the printer and video handlers use it for temporary storage.

Next month this column will discuss methods for sorting disk files using this program.

The assembly language and Basic code is listed on pages 2, 3, & 4. The programs are also contained in MUG Library Disk 30, for those of you who don't want to type it all in.

.....

BASIC SUBROUTINES

by Buzz Rudow

The use of subroutines is a subject we've touched on before (see newsletter 2, page 2). Since BASIC/Z is a compiler, rather than an interpreter, I decided to do a bit of re-evaluation of the use of subroutines.

These tests were run with Micropolis Basic as the interpretive, and BASIC/Z as the compiler standard. However, to the best of my knowledge, the results hold generally true for all languages of their type. That is, for the most part, BASIC-80, CBASIC, UCSD Pascal, Pascal/M and other interpretative languages work like Micropolis Basic. BASCOM (compiled BASIC-80), CB80 (compiled CBASIC), most "C" languages, FORTRAN, and PASCAL/Z work like BASIC/Z.

(Continued on page 5)

Title: SORT.ASM

```
0000 * **** SORT.ASM ************************************************************
0010 *
0020 * COPYRIGHT 1981 BY BURKS A. SMITH
0030 * RELEASED FOR THE PRIVATE USE OF MICROPOLIS USERS GROUP.
0040 * SALE PROHIBITED WITHOUT WRITTEN PERMISSION OF THE AUTHOR.
0050 *
0060 ************************************************************************
0070 *
0080 * WRITTEN BY BURKS A. SMITH 4-15-81
0090 *             DATASMITH
0100 *             BOX 8036
0110 *             SHAWNEE MISSION KS 66208
0120 * MOD 10-8-81
0130 *
0140 ****************************************************
0150 * SUBROUTINE EQUATES  MDOS VS. 4.0                 *
0160 ****************************************************
0170 ARG1            EQU    04BCH          ;ARGUMENT 1
0180 ARG2            EQU    04BEH          ;ARGUMENT 2
0190 ARG3            EQU    04C0H          ;ARGUMENT 3
0200 NARGS           EQU    04C4H          ;# OF ARGUMENTS
0210 RSIZE           EQU    04C5H          ;REAL PRECISION
0220 ISIZE           EQU    04C6H          ;INTEGER PRECISION
0230 SSIZE           EQU    04C7H          ;STRING PRECISION
0240 RESULT          EQU    01A0H          ;RESULT
0250 @CBRK           EQU    785H           ;BREAK CHECK
0260 @DISKERROR      EQU    1C8FH          ;ERROR ABORT ADDRESS
0270 *
0280 * MICROPOLIS BASIC VS. 4.0 DATA ARRAY POINTERS
0290 *
0300 AARRAY          EQU    33B9H          ;START OF A$ ARRAY
0310 *
0320 ************************************************************************
0330                 ORG    0BE00H         ;ASSY ADDRESS **** 48K VERSION ****
0340 ************************************************************************
0350 *
0360 *
0370 BEGIN           JMP    SORTENTRY      ;SORT ENTRY POINT
0380                 NOP                   ;JUST A PLACEHOLDER
0390 *
0400 * THE FOLLOWING MUST BE POKED BEFORE EXECUTION
0410 *
0420 SIZE            DW     00             ;LAST INDEX TO SORT+1
0430 KEY1            DB     0              ;KEY 1 STARTING ADDR
0440                 DB     0              ;KEY 1 LENGTH TO CHECK
0450                 DB     0              ;KEY 1 SEQUENCE (0=ASC, 1=DEC)
0460 KEY2            DB     0,0,0          ;KEY 2 START,LEN,SEQ
0470 KEY3            DB     0,0,0
0480 KEY4            DB     0,0,0
0490 KEY5            DB     0,0,0
0500 KEY6            DB     0,0,0
0510 KEY7            DB     0,0,0
0520 KEY8            DB     0,0,0
0530 KEY9            DB     0,0,0
0540 KEY10           DB     0,0,0
0550                 DB     0              ;MUST ALWAYS BE ZERO
0560 *
0570 * TEMPORARY STORAGE
```


Title: SORT.ASM

```
0580 *
0590 START           DB     0              ;CURRENT KEY START
0600 LEN             DB     0              ;CURRENT KEY LEN
0610 SEQ             DB     0              ;CURRENT KEY SEQUENCE
0620 LASTKEY         DW     0              ;KEY INFO ADDRESS
0630 SWAPFLAG        DB     0              ;SWAP INDICATOR
0640 STRLEN          DB     0              ;STRING LENGTH
0650 INDEX1          DW     00             ;SHELL BOTTOM INDEX
0660 INDEX2          DW     00             ;SHELL TOP INDEX
0670 ADDR1           DW     00             ;ADDRESS OF START OF DATA
0680 *
0690 ****************************************************
0700 *  SORT ROUTINE
0710 ****************************************************
0720 *
0730 * SORT KEYS ARE PASSED TO ROUTINE WITH POKE COMMANDS.
0740 * THIS PROGRAM EXPECTS TO FIND DATA TO BE SORTED IN ARRAY A$
0750 * AND RETURNS ARRAY A$ IN SORTED CONDITON.
0760 * IF AN ERROR OCCURS, THE PROGRAM RETURNS THE CHARACTER 'E'.
0770 * IF THE SORT IS SUCCESSFUL, THE PROGRAM RETURNS THE CHARACTER 'A'.
0780 ****************************************************
0790 SORTENTRY       LHLD   AARRAY         ;GET START ADDRESS
0800                 MOV    A,M            ;GET # OF DIMENSIONS
0810                 CPI    1              ;MUST BE 1
0820                 JNZ    SERROR         ;ELSE ERROR
0830                 INX    H              ;POINT TO # OF ELEMENTS
0840                 INX    H              ;(IGNORE IN THIS VERSION)
0850                 XCHG
0860                 LHLD   SIZE           ;LOAD # OF ELEMENTS
0870                 SHLD   INDEX2         ;PRIME SHELL
0880                 XCHG
0890                 INX    H              ;POINT TO LEN
0900                 MOV    A,M            ;GET LEN
0910                 STA    STRLEN         ;SAVE
0920                 INX    H              ;1ST CHAR
0930                 SHLD   ADDR1          ;SAVE START OF ARRAY
0940 *
0950 SORTLOOP        CALL   SETINDEX       ;CALCULATE SHELL INDEX
0960                 LXI    D,0            ;ZERO FOR COMPARISON
0970                 CALL   COMPARE
0980                 JZ     SEXIT          ;NORMAL EXIT IF HL=0
0990 SORTLOOP1       LXI    H,0            ;00 TO HL
1000                 SHLD   INDEX1         ;INIT START
1010                 XRA    A              ;MAKE A ZERO
1020                 STA    SWAPFLAG       ;INIT SWAP FLAG
1030                 LHLD   INDEX1         ;BOTTOM OF SHELL
1040                 XCHG                  ;TO DE
1050                 LHLD   INDEX2         ;TOP OF SHELL TO HL
1060 SORTLOOP2       CALL   RESET          ;SET TO FIRST KEY
1070                 CALL   @CBRK          ;CHECK FOR ^C
1080                 JZ     @DISKERROR     ;EXIT IF SO
1090                 CALL   SWAP           ;SWAP IF NEEDED
1100                 INX    H              ;BUMP TOP
1110                 INX    D              ;AND BOTTOM
1120                 PUSH   D              ;SAVE BOTTOM
1130                 XCHG
1140                 LHLD   SIZE           ;# OF ELEMENTS
1150                 XCHG
```

Title: SORT.ASM

```
1160                CALL    COMPARE         ;SEE IF DONE
1170                POP     D               ;RESTORE
1180                JNZ     SORTLOOP2       ;LOOP IF NOT
1190 * SHELL PASS COMPLETE
1200                LDA     SWAPFLAG        ;GET SWAP FLAG
1210                ORA     A               ;SET FLAGS
1220                JNZ     SORTLOOP1       ;ANOTHER PASS NEEDED IF NZ
1230                JMP     SORTLOOP        ;ELSE DECR SHELL
1240 *
1250 * DIVIDE SHELL INDEX BY TWO
1260 *
1270 SETINDEX       LHLD    INDEX2          ;GET INDEX
1280                MOV     A,H             ;HIGH BYTE
1290                ORA     A               ;CLEAR CARRY
1300                RAR                     ;DIVIDE BY 2
1310                MOV     H,A             ;REPLACE
1320                MOV     A,L             ;LOW BYTE
1330                RAR                     ;DIVIDE BY 2 & SHIFT CARRY
1340                MOV     L,A             ;REPLACE
1350                SHLD    INDEX2          ;BACK IN MEMORY
1360                RET                     ;AND RETURN
1370 *
1380 * COMPARE TWO STRINGS
1390 *
1400 SWAP           PUSH    D               ;SAVE BOTTOM
1410                PUSH    H               ;AND TOP
1420 * FIRST FIND ADDRESSES OF STRINGS TO COMPARE
1430                LHLD    ADDR1           ;ARRAY START
1440                LDA     STRLEN          ;LENGTH OF EA STRING
1450                MOV     B,A             ;TO B
1460                CALL    INCREMENT       ;POINT TO ELEMENT
1470                POP     D               ;GET TOP BACK
1480                PUSH    D               ;BUT LEAVE ON STACK
1490                PUSH    H               ;SAVE BOTTOM ADDR
1500                LHLD    ADDR1           ;ARRAY START
1510                LDA     STRLEN          ;STRING LENGTH
1520                MOV     B,A             ;TO B
1530                CALL    INCREMENT       ;POINT TO ELEMENT
1540                POP     D               ;BOTTOM ADDR IN DE -TOP IN HL
1550                CALL    KEYCOMP         ;COMPARE THE TWO STRING
1560                JZ      SWAPRET         ;RETURN IF EQUAL
1570 * UNEQUAL COMPARISON
1580 SWAP1          JNC     SWAP2           ;DECENDING COMPARE JUMP
1590                LDA     SEQ             ;GET SEQUENCE
1600                CPI     01              ;DECENDING SORT?
1610                JZ      EXCHANGE        ;SWAP IF SO
1620                JMP     SWAPRET         ;NORMAL EXIT
1630 SWAP2          LDA     SEQ             ;GET SEQUENCE
1640                CPI     00              ;ASCENDING SORT?
1650                JZ      EXCHANGE        ;SWAP IF SO
1660 *
1670 SWAPRET        POP     H               ;TOP INDEX
1680                POP     D               ;BOTTOM INDEX
1690                RET                     ;NORMAL RETURN
1700 *
1710 * EXCHANGE TWO STRINGS
1720 *
1730 EXCHANGE       LDA     STRLEN          ;LENGTH OF STRING
```


Title: SORT.ASM

```
1740                MOV     B,A             ;TO B
1750                DCR     B               ;SET UP FOR LOOP
1760 EXCH1          LDAX    D               ;BOTTOM
1770                MOV     C,A             ;SAVE TEMPORARILY
1780                MOV     A,M             ;TOP
1790                STAX    D               ;TO BOTTOM
1800                MOV     M,C             ;BOTTOM TO TOP
1810                INX     H               ;BUMP ADDRESSES
1820                INX     D
1830                DCR     B               ;AND COUNTER
1840                JNZ     EXCH1           ;LOOP IF NEEDED
1850                MVI     A,01            ;MAKE A 1
1860                STA     SWAPFLAG        ;AND SET FLAG
1870                JMP     SWAPRET         ;OR RETURN
1880 *
1890 * COMPARE DE AND HL
1900 *
1910 COMPARE        MOV     A,D             ;GET D
1920                CMP     H               ;COMPARE WITH H
1930                RNZ                     ;RETURN NOT EQUAL
1940                MOV     A,E             ;ELSE GET E
1950                CMP     L               ;COMPARE WITH L
1960                RET                     ;AND RETURN
1970 *
1980 * INCREMENT TO START OF STRING
1990 *
2000 INCREMENT      DAD     D               ;INDEX
2010                DCR     B               ;DECREMENT COUNTER
2020                JNZ     INCREMENT       ;CONTINUE TILL DONE
2030                INX     H               ;LENGTH (IGNORE)
2040                RET                     ;NORMAL RETURN
2050 *
2060 * EXIT ROUTINES
2070 *
2080 SERROR         LXI     H,RESULT+3      ;RETURN FLAG
2090                MVI     M,'E'           ;ERROR INDICATOR
2100                JMP     SEXIT1          ;AND GO
2110 *
2120 SEXIT          LXI     H,RESULT+3      ;RETURN FLAG
2130                MVI     M,'A'           ;OK INDICATOR
2140 SEXIT1         DCX     H               ;CURRENT LEN
2150                MVI     M,01
2160                DCX     H               ;MAX LEN
2170                MVI     M,01
2180                DCX     H               ;TYPE
2190                MVI     M,03            ;STRING TYPE
2200                RET                     ;GENERAL EXIT
2210 *
2220 * RESET TO FIRST KEY
2230 *
2240 RESET          PUSH    H               ;SAVE REGISTER
2250                LXI     H,KEY1          ;POINT TO FIRST KEY
2260                CALL    NEWKEY          ;GET THE KEY
2270                POP     H               ;RESTORE REGISTER
2280                RET                     ;EXIT
2290 *
2300 * SAVE NEW KEY PARAMETERS
2310 *
```

Title: SORT.ASM

```
2320 NEWKEY          MOV     A,M             ;START INDEX
2330                 STA     START
2340                 INX     H
2350                 MOV     A,M             ;LENGTH INDEX
2360                 STA     LEN
2370                 INX     H
2380                 MOV     A,M             ;SEQUENCE INDEX
2390                 STA     SEQ
2400                 SHLD    LASTKEY         ;SAVE ADDRESS
2410                 RET                     ;AND RETURN
2420 *
2430 * COMPARE TWO STRINGS BY KEY DATA
2440 *
2450 * RETURNS ZERO SET IF COMPARISON EQUAL
2460 * RETURNS C IF 'ASCENDING'
2470 * RETURNS NC IF 'DECENDNG'
2480 *
2490 KEYCOMP         LDA     START           ;LOAD KEY START INDEX
2500                 PUSH    D               ;SAVE STRING START ADDRESSES
2510                 PUSH    H
2520 KEYCOMP1        INX     H               ;BUMP ADDRESSES
2530                 INX     D
2540                 DCR     A               ;DECREMENT COUNT
2550                 JNZ     KEYCOMP1        ;UNTIL POINTING AT KEY FIELDS
2560 * DO COMPARISON
2570                 LDA     LEN             ;LENGTH TO COMPARE
2580                 MOV     B,A             ;TO B
2590 KEYCOMP2        LDAX    D               ;COMPARE BYTE OF EACH FIELD
2600                 CMP     M
2610                 JZ      KEYCOMP3        ;CONTINUE COMPARISONS IF =
2620 KEYRET          POP     H               ;ELSE RESTORE POINTERS
2630                 POP     D
2640                 RET                     ;AND RETURN
2650 *
2660 KEYCOMP3        DCR     B               ;DECR CHAR COUNT
2670                 JZ      NEXTKEY         ;GET NEW KEY IF DONE WITH THIS ONE
2680                 INX     D               ;ELSE BUMP POINTERS
2690                 INX     H
2700                 JMP     KEYCOMP2        ;AND LOOP
2710 *
2720 * POINT TO NEXT KEY
2730 *
2740 NEXTKEY         LHLD    LASTKEY         ;GET LAST KEY ADDR
2750                 INX     H               ;POINT TO NEXT KEY START
2760                 MOV     A,M             ;FETCH THE BYTE
2770                 ORA     A               ;SET FLAGS
2780                 JZ      KEYRET          ;DONE IF LAST KEY
2790                 CALL    NEWKEY          ;ELSE GET THE KEY INFO
2800                 POP     H               ;RESTORE STRING START ADDRESSES
2810                 POP     D
2820                 JMP     KEYCOMP         ;REPEAT FOR NEXT KEY
2830 *
2840                 END     BEGIN
```

Title: ASORT.BAS

```
  10 ! **** ASORT ****    Written by Burks A. Smith
  15 !
  20 ! BASIC PROGRAM TO USE ASSEMBLY LANGUAGE SORT OF A$
  25 ! DATA IS EXPECTED TO BE IN FIXED FIELD LENGTH FORMAT.
  30 ! ARRAY K% HOLDS START POSITION AND LENGTH OF ALL KEYS.
  35 !
  40 ! A$( ) - ARRAY HOLDING STRINGS TO SORT
  45 ! K%(X,0) - KEY X START
  50 ! K%(X,1) - KEY X LENGTH
  55 ! K%(X,2) - KEY X SEQUENCE 0=ASCENDING, 1=DECENDING
  60 ! I% - NUMBER OF KEYS
  65 !
 100 MEMEND 16RDDFF:  !LEAVE SPACE FOR SUBROUTINE
 110 DEF FAS=16RDE00: !DEFINE ENTRY POINT
 120 M=16RDE04:       !ADDRESS OF DATA AREA FOR PARAMETERS.
 130 LOAD "SORTPGM":  !LOAD SUBROUTINE
 140 !
 200 ! SAMPLE PROGRAM TO SORT A DATA FILE
 210 OPEN 1 "DATAFILE" END 300
 220 DIM A$(SIZE(1),250), K%(2,2)
 230 N%=0
 240 GET 1 L$,F$,N
 250 A$(N%)=LEFT$(L$+REPEAT$(" ",20),20): !FIX LENGTH
 260 A$(N%)=A$(N%)+LEFT$(F$+REPEAT$(" ",20),20)
 270 A$(N%)=A$(N%)+FMT(N,"ZZZZZ")
 280 N%=N%+1
 290 GOTO 240
 300 N%=N%-1: I%=2
 310 K%(1,0)=1: K%(1,1)=20: K%(1,2)=0 :!KEY 1
 320 K%(2,0)=21: K%(2,1)=20: K%(2,2)=0: !KEY 2
 330 GOSUB 1000: !SORT
 340 !< PGM USES SORTED DATA >
 350 CLOSE 1
 360 STOP
 370 !
1000 !< SORT ARRAY A$ SUBROUTINE
1010 !
1020 POKE(M)=(N%+1) AND 16R00FF: !LOW ORDER SIZE TO SORT BYTE
1030 POKE(M+1)=((N%+1) AND 16RFF00)\16RFF: !HIGH ORDER SIZE
1040 M=M+2
1050 FOR X%=1 TO I%: !POKE IN ALL KEYS
1060 POKE(M)=K%(X%,0)
1070 POKE(M+1)=K%(X%,1)
1080 POKE(M+2)=K%(X%,2)
1090 M=M+3
1100 NEXT X%
1110 POKE(M)=0: !END OF KEYS MARK
1120 A$=FAS :   !SORT THE FILE
1130 IF A$="E" THEN PRINT "ERROR": STOP
1140 RETURN
```

Suppose you have a routine that is used more than once in your program. I have one that I use after I've displayed something to the screen. I'm waiting for the user to review the material and tell the program to continue. It goes like this.

```
010 DIM R$(1)
    ...
100 R$=""
110 PRINT
120 INPUT "When Ready to Continue, Type: 'C<RETURN>
    '";R$
130 IF R$<>"C" THEN GOTO 100
    ...
200 R$=""
210 PRINT
220 INPUT "When Ready to Continue, Type: 'C<RETURN>
    '";R$
230 IF R$<>"C" THEN GOTO 200
```

In interpretive Basics, the rewriting of this code each time you use it costs you memory space equal to the sum of the number of characters the routine takes. Generally, this is four bytes for each line number, and one byte for each other character or space. The exception is that key-words are tokenized. That is, "PRINT", "INPUT", etc., each take only one byte.

Writing one of the above interpretive routines as a subroutine saves 60 bytes. The code looks like the following.

```
010 DIM R$(1)
    ...
100 GOSUB 500
    ...
200 GOSUB 500
    ...
500 R$=""
510 PRINT
520 INPUT "When Ready to Continue, Type: 'C<RETURN>
    '";R$
530 IF R$<>"C" THEN GOTO 500
540 RETURN
```

What you lose is some speed of execution. The interpreter must find the subroutine. In Micropolis Basic, this lost time is directly proportional to the location of the routine in the program. For any GOSUB (or GOTO), Micropolis BASIC goes back to the front of the program and looks through each byte of code until it finds the proper line number. It pays to have subroutines which are executed repetitively, be up front in your program.

So, for interpretive languages, using subroutines gains you approximately a byte for byte savings of memory space, and loses an indeterminate, and often sizable, amount of execution speed.

The code I use for the same function looks somewhat different in BASIC/Z, but I'm sure you can see it's the same result. I could have used R$, but I tend to make my variable more descriptive.

```
010 DIM RESPONSE$(1)
    ...
100 RESPONSE$=""
110 PRINT
120 INPUT "When Ready to Continue, Type: 'C<RETURN>
    '";RESPONSE$
130 IF RESPONSE$<>"C" THEN GOTO 100
    ...
200 RESPONSE$=""
210 PRINT
220 INPUT "When Ready to Continue, Type: 'C<RETURN>
    '";RESPONSE$
230 IF RESPONSE$<>"C" THEN GOTO 200
```

A compiling language generates the total code required to perform the routine. The lengthy variable names do not impact compiled code size, since they are transcribed to a memory location number, regardless of the length of the name. The length of the code generated for each routine should be much larger, however. This is because the code will execute in line, rather than having the interpreter do the bulk of the execution. Rewriting of the code should prove very expensive in terms of memory space.

Writing the above compiling routines as a subroutine saves 56 bytes. I was somewhat surprised that the savings weren't greater. I suppose it has something to do with the BASIC/Z compiled code being calls to routines in the RUN/Z module, rather than pure in-line code. The code looks like the following.

```
010 DIM RESPONSE$(1)
    ...
100 GOSUB @WAIT.TO.CONTINUE
    ...
200 GOSUB @WAIT.TO.CONTINUE
    ...
490 @WAIT.TO.CONTINUE
500 RESPONSE$=""
510 PRINT
520 INPUT "When Ready to Continue, Type: 'C<RETURN>
    '";RESPONSE$
530 IF RESPONSE$<>"C" THEN GOTO @WAIT.TO.CONTINUE
540 RETURN
```

You still lose a bit of execution speed. Not much though. Just the time to do a machine language CALL to a memory location and a RETURN - a couple of milliseconds.

For compiling languages, then, using subroutines gains you a variable number of bytes of memory space with the loss of a very small, fixed, amount of execution time.

From my point of view, given the limited memory space in our micro-computers, the use of subroutines in an interpretive language is generally an advantage, if you watch out where you put them. When using a compiling language, the use of subroutines is almost a must.

It was interesting to find out that the repetitive setting of any variable could be constructed as a subroutine in BASIC/Z with a resultant saving of memory space. Consider the following two subroutines.

```
10 ! Test 1              10 ! Test 2
20 R$=""                 20 GOSUB 50
30 INPUT R$              30 INPUT R$
40 RETURN                40 RETURN
50 R$=""                 50 R$=""
60 RETURN                60 RETURN
```

In BASIC/Z, Test 2 takes 6 bytes less memory space than Test 1. In Micropolis Basic, Test 2 takes 1 byte less space than Test 1. The same result holds true whether you're setting a string variable like "R$", or a real number like "T".

A last little tid-bit of information is about "structured programming". It seems that BASIC/Z allows one to write structured code without losing memory space. The following code is a structured version of @WAIT.TO.CONTINUE which takes 3 bytes less code than the former version.

```
010 DIM RESPONSE$(1)
    ...
100 GOSUB @WAIT.TO.CONTINUE
    ...
200 GOSUB @WAIT.TO.CONTINUE
    ...
490 @WAIT.TO.CONTINUE
500 DO
510     RESPONSE$=""
520     PRINT
530     INPUT "When Ready to Continue, Type: 'C<RET
        URN>'";RESPONSE$
540 UNTIL RESPONSE$="C"
550 RETURN
```

.....

NEW AREAS OF ATTENTION

by Buzz Rudow

Special Interest Groups (SIGs)

Most general purpose computer clubs have SIGs to support areas of hardware and software interest. The MUG has always had areas which received more attention than others, but nothing has been formalized. The areas that I find interesting are certainly not the same as what interests each of you. To start, I've established two SIGs; one hardware, one software.

Vector Graphic

The hardware SIG is for Vector Graphic systems. A large percentage of the MUG members have always been Vector Graphic owners. The percentage is rising. VG owners have more interest in their versions (dozens of versions) of CP/M, of monitors, of VG software, and in the setup and use of the Flashwriter and Bitstreamer boards.

A monthly column will cover areas of interest to the VG owners. Exactly what is printed is up to the VG membership. You people have to supply the material. At the moment, there is no groups leader for the VG. Send your material, questions, and thoughts for direction to me. If you wish to be responsible for the monthly column, drop me a line. The VG column starts this month with an article by Herbert Spirer.

Basic/z

A second area of monthly attention will be Basic/z. Basic/z is truly a fine package. A large number of our members have bought either the MDOS or the CP/M version. A few have bought both. Being a fine package isn't the same as saying it's simple to use all of Basic/z' power. Steve Guralnick has committed himself to writing the Basic/z column, which will start in February. You can contact Steve at 375 S. Mayfair, Suite 205, Daly City CA 94015, or call him at 415/992-9200 (days) or 415/991-0155 (nights). Let him know what you'd like to see in his articles. Send him articles of your own that can be used, or contribute bits of information you've learned which can be mentioned by Steve.

Anymore?

The above two areas are certainly not the total of special interests of the MUG membership. There are a large number of Exidy and SOL owners, but they have external clubs which cover their hardware. The COMPAL system is a possible candidate, however, and perhaps CDS. From what I hear, COMPAL has some nice system software, i.e., enhancements of, or inovative use of, the MDOS package. Some of you may wish to establish SIGs for Basic-80 or C-Basic.

I'm willing to establish as many SIGs as you wish. The "you" is the key. I'm willing to provide the space in the newsletter, but am not able to provide the special interest material. Write or call if you wish to discuss the degree of MUG interest in your particular special interest.

.....

GETTING UPDATES FOR YOUR LIBRARY DISKS

The new format of the library has disks being established for an area, such as games, or home. When the disk is first released, it generally is not "full" - full being a MOD I worth. Members buy this disk for $3 and a program submitted on their disk, or for $10 outright. If I add another program to the disk, it is unfair for me to charge another program submittal or another $10 for that update.

The MUG policy, therefore, is that updates to any previously purchased library disk can be had for $3, if you supply the disk. No additional program need be submitted.

.....

****
* *
* VECTOR GRAPHIC SIG *
* *
****

VECTOR GRAPHIC TIPS I

By Herbert F. Spirer
University of Connecticut, Stamford, CT

(NOTE: These comments refer to my experience with a Vector Graphic System B, CP/M 2.2.)

COMMUNICATING, CONTROL-E:

In MUG Newsletter #27, page 8, Susan Kleinman reports on difficulty BREAKing and gave the right fix, to insert control-E after the automatic load query when running CONFIG. She also said that this fix "is undocumented in Vector literature." However, it is documented. On page 12 of the Vector CONECT Users Manual, Revision B, September 17, 1982, Vector instructs the user to carry out the same action as Susan recommends.

COMMUNICATING, RECEIVING INTO A FILE:

In the INTERACTIVE mode, CONECT allows you to put incoming data into a file. The file name is defined by the user by executing control-Y and entering disk identification, file-name and extension in a predefined field which looks like this:

A:--------.---

The user is not warned that the file name must be left-, not right-justified.

If you enter a file-name as

A:_ T E L E C O M . M E M

you will be unable to retrieve it using standard CP/M file handling (which includes MEMORITE). As is clear from examination of the directory, CONECT will have created a file-name with a leading blank and the file is inaccessible if you request it as TELECOM.MEM. Also, since leading spaces in your keyboard entries for file access are ignored, you will not be able to simply prefix a space.

To recover such a file, use the wild card "?" for the inserted blanks, provided this does not create an ambiguity. Thus,

TYPE TELECOM.MEM

will not type the file, but

TYPE ?TELECOM.MEM

will.

It is good practice to use a unique file-name structure for communications files so that you can always recover them should you accidentally insert a space.

CIVILIZING MEMORITE FILES:

MEMORITE files are random, and for this reason are not accessible to SCOPE and not directly SENDable by telecommunications. In fact, the Vector SCOPE

(Continued on Page 10)

PGMNAME/TOPIC	CO./AUTHOR	PGM/ARTICLE TYPE	CATEGORY	VOL-PG
A-FORTH	ACROPOLIS	HIGH LEVEL LANGUAGE	COMPILER	012-03
A-FORTH COMPILER	ACROPOLIS	HIGH LEVEL LANGUAGE	COMPILER	020-02
A-FORTH PATCHES		HIGH LEVEL LANGUAGE	COMPILER	024-19
ACCESSING DISK FILES*	B.RUDOW	BASIC PGM TECHNIQUE	DISK FILES	020-03
ACROPOLIS UTILITIES	ACROPOLIS	8080 UTILITY PGM		022-14
ACROPOLIS UTILITIES	ACROPOLIS	8080 UTILITY PGM		023-03
AMORT	B.SMITH	BASIC APPL PGM	BUSINESS	008-05
ASCII MEMORY DUMP	ACROPOLIS	8080 UTILITY PGM		023-05
ASMTEXT*	B.RUDOW	8080 APPL PGM	TERM I/O	005-05
ASSEMBLER PGMING BOOKS		8080 PGMING	REFERENCE	008-04
ASSEMBLER PROGRAMMING*		8080 PGMING		005-01
AUTO-CONFIGURATION*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	019-08
BANKING PGM		8080 APPL PGM	BUSINESS	012-05
BAS>LIN	DATASMITH	8080 UTILITY PGM		024-03
BASIC BUGS		BASIC DOC	REFERENCE	022-01
BASIC LOAD+GO	MICROPOLIS	8080 UTILITY PGM		006-10
BASIC PGMING		BASIC PGMING		012-04
BASIC PGMING		BASIC PGMING		013-01
BASIC PGMING		BASIC PGMING		015-01
BASIC PGMING		BASIC PGMING		016-01
BASIC PGMING BOOKS		BASIC PGMING	REFERENCE	012-06
BASIC PGMING*	B.SMITH	BASIC PGMING		014-01
BASIC TOKENS		BASIC DOC	REFERENCE	009-03
BASIC TOKENS		BASIC DOC	REFERENCE	018-02
BASIC TOKENS*	W.POWERS	BASIC DOC	REFERENCE	014-05
BASIC TOKENS/KEYWORDS		BASIC DOC	REFERENCE	020-13
BASIC UTILITIES	GMS	8080 UTILITY PGM		023-10
BASIC VARIABLE POINTERS		BASIC DOC	REFERENCE	021-01
BASIC/S + BASIC/Z	SYSTEMATION	HIGH LEVEL LANG	COMPILER	022-06
BASIC/S + BASIC/Z	SYSTEMATION	HIGH LEVEL LANGUAGE	COMPILER	020-06
BASIC/S COMPILER	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	001-01
BASIC/S COMPILER	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	006-06
BASIC/S COMPILER	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	010-06
BASIC/S COMPILER	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	012-01
BASIC/S COMPILER	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	013-01
BASIC/S COMPILER*	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	014-01
BASPAC	GMS	8080 UTILITY PGM		023-12
BATCHCOPY*	C.SINGER	8080 UTILITY PGM	DISK FILES	017-06
BLACKHAWK COMPUTER		HARDWARE		022-07
BOOKKEEPING	DATASMITH	BASIC APPL PGM	BUSINESS	006-03
BOOKKEEPING	DATASMITH	BASIC APPL PGM	BUSINESS	020-09
BSS	INV ANAL SYS	BASIC APPL PGM	BUSINESS	008-03
CCA DMS	CUSTOM ELEC	BASIC APPL PGM	DATA BASE	003-07
CCA DMS	CUSTOM ELEC	BASIC APPL PGM	DATA BASE	004-13
CDS VERSATILE COMPUTER	CDS	HARDWARE		014-02
CHEAP COMPUTER		HUMOUR		022-02
CHEAP COMPUTER		HUMOUR		023-02
CHEAP COMPUTER		HUMOUR		024-02
CLASSIFIED ADS		GENERAL INFO		009-06
CLASSIFIED ADS		GENERAL INFO		011-06
CLASSIFIED ADS		GENERAL INFO		012-06
CLASSIFIED ADS		GENERAL INFO		013-06
CLASSIFIED ADS		GENERAL INFO		014-08
CLASSIFIED ADS		GENERAL INFO		015-08
CLASSIFIED ADS		GENERAL INFO		016-16
CLASSIFIED ADS		GENERAL INFO		017-16
CLASSIFIED ADS		GENERAL INFO		018-16
CLASSIFIED ADS		GENERAL INFO		019-12
CLASSIFIED ADS		GENERAL INFO		020-14
CLASSIFIED ADS		GENERAL INFO		021-15
CLASSIFIED ADS		GENERAL INFO		022-15
CLASSIFIED ADS		GENERAL INFO		024-20
CLEAR SCREEN		BASIC PGM TECHNIQUE	TERM I/O	006-06
CLEAR SCREEN*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	001-02
CLEAR SCREEN*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	002-01
CLEAR SCREEN*	J.CALLAWAY	BASIC PGM TECHNIQUE	TERM I/O	006-05
COMMUNICATIONS		8080 SYSTEM PGM		018-03
COMPAL-8200		HARDWARE		016-11
COMPRESS	ACROPOLIS	8080 UTILITY PGM		023-06
COMPUTER NO'S		PGMING THEORY		017-01
CONTROL-P		8080 SYSTEM PGM	OP SYSTEM	018-08
CONTROL-P*	B.CARIGNAN	8080 SYSTEM PGM	OP SYSTEM	018-12
CP/M		8080 SYSTEM PGM	OP SYSTEM	004-06
CP/M		8080 SYSTEM PGM	OP SYSTEM	006-01
CP/M		8080 SYSTEM PGM	OP SYSTEM	006-02
CP/M		8080 SYSTEM PGM	OP SYSTEM	008-05
CP/M		8080 SYSTEM PGM	OP SYSTEM	011-02
CP/M		8080 SYSTEM PGM	OP SYSTEM	012-03
CP/M		8080 SYSTEM PGM	OP SYSTEM	013-02
CP/M		8080 SYSTEM PGM	OP SYSTEM	014-02
CP/M - MDOS COMPARISON		8080 SYSTEM PGM	OP SYSTEM	014-06
CP/M - MDOS COMPARISON		8080 SYSTEM PGM	OP SYSTEM	024-01
CP/M - MDOS CONVERSION*	B.RUDOW	UTILITY PGM		019-06
CP/M HANG FIXES		8080 SYSTEM PGM	OP SYSTEM	020-12
CP/M*	S.TATTERSALL	8080 SYSTEM PGM	OP SYSTEM	010-05
CRUNCH	SYSTEMATION	BASIC PGMING AID		004-09
CRUNCH	SYSTEMATION	BASIC PGMING AID		008-07
CURSOR CONTROLS*	DAVE LAND	BASIC PGM TECHNIQUE	TERM I/O	008-01
CURSOR CONTROLS*	J.FACTOR	BASIC PGM TECHNIQUE	TERM I/O	013-03
DATABASE	J.SHAPIRO	BASIC APPL PGM	DATA BASE	003-07
DATABASE	BONJOEL	BASIC APPL PGM	DATA BASE	005-01
DATABASE TWO	BONJOEL	BASIC APPL PGM	DATA BASE	006-07
DATABASE TWO	BONJOEL	BASIC APPL PGM	DATA BASE	006-08
DATE ACCESSES*	D.O'BRIEN	BASIC PGM TECHNIQUE		009-03
DATE ACCESSES*	ED BURKHARDT	BASIC PGM TECHNIQUE		012-05
DEBUG		8080 SYSTEM PGM	DEBUGGER	017-04
DFILE	DATASMITH	8080 UTILITY PGM		024-07
DIM		BASIC DOC	REFERENCE	019-01
DISASSEMBLER	CUSTOM ELEC	8080 SYSTEM PGM	DISASSM	003-07
DISK BANKING	J.LENZ	BASIC APPL PGM	BUSINESS	022-11
DISK CATALOG SYSTEM*	B.RUDOW	BASIC UTILITY PGM	DISK DIR	017-10
DISK DRIVE BELTS		HARDWARE		016-13
DISK DRIVE SALE	PRIORITY ONE	HARDWARE		004-12
DISK ERRORS		DISK MEDIA		014-06
DISK FILE ACCESS		BASIC PGM TECHNIQUE	DISK FILES	004-05
DISK FILE ACCESSES*	B.RUDOW	BASIC PGM TECHNIQUE	DISK FILES	021-03
DISK FILE ACCESSES*	B.RUDOW	BASIC PGM TECHNIQUE	DISK FILES	024-05
DISK HUB RINGS		HARDWARE		023-14
DISK I/O ERRORS		DISK MEDIA		005-10
DISK MAINTENANCE		HARDWARE		023-13
DISK RUDIMENTS		DISK MEDIA		010-04
DISK RUDIMENTS		DISK MEDIA		011-03
DISKDUMP	DATASMITH	8080 UTILITY PGM		024-07
DMEM	DATASMITH	8080 UTILITY PGM		024-07
DOCGEN		BASIC APPL PGM	WORD PROC	018-09
DOUBLING KEYS		8080 SYSTEM PGM	OP SYSTEM	018-08
DRIVE TURN-OFF		HARDWARE		016-03
EXECUTION TIME*	B.RUDOW	BASIC PGM TECHNIQUE		002-02

MICROPOLIS USER'S GROUP NEWSLETTER INDEX - CUMULATIVE YEARS 1 & 2

PGMNAME/TOPIC	CO./AUTHOR	PGM/ARTICLE TYPE	CATEGORY	VOL-PG
EXECUTION TIME*	B.RUDOW	BASIC PGM TECHNIQUE		012-04
FILE OPEN ROUTINE*	B.SMITH	BASIC PGM TECHNIQUE	DISK FILES	009-04
FINANCIAL PLANNING PGMS	SYNTAX CORP.	BASIC APPL PGM	BUSINESS	006-04
FLASHWRITER II	VECTOR GR	HARDWARE		011-01
FLIPPY DISKS	MICRO-SERVE	HARDWARE		012-06
FLIST	GMS	8080 UTILITY PGM		023-08
FMT		BASIC DOC	REFERENCE	009-03
FMT*	B.SMITH	BASIC DOC	REFERENCE	012-01
FORMATTED INPUT*	J.FACTOR	BASIC PGM TECHNIQUE	TERM I/O	022-05
FORTH		HIGH LEVEL LANGUAGE	COMPILER	013-02
FORTH		HIGH LEVEL LANGUAGE	COMPILER	013-03
FORTH		HIGH LEVEL LANGUAGE	COMPILER	023-16
FORTH CONCEPTS		HIGH LEVEL LANGUAGE	COMPILER	024-04
FORTH*	R.NEWMAN	HIGH LEVEL LANGUAGE	COMPILER	021-02
FORTH*	R.NEWMAN	HIGH LEVEL LANGUAGE	COMPILER	022-03
GENERAL LEDGER PGMS	MODERN MICRO	BASIC APPL PGM	BUSINESS	013-03
GENSORT*	ED BURKHARDT	BASIC UTILITY PGM	SORT	007-01
GET-TRAX		8080 UTILITY PGM	DISK FILES	018-06
GOTO*	B.RUDOW	BASIC DOC	REFERENCE	002-02
GRAPHICS		BASIC PGM TECHNIQUE		010-01
GRAPHICS*	B.SMITH	BASIC PGM TECHNIQUE		011-01
GRAPHICS*	A.PICKERT	BASIC PGM TECHNIQUE		016-02
HAM PGMS AVAILABLE		8080 APPL PGM	HAM RADIO	005-12
HIGH LEVEL LANGUAGES		GENERAL INFO		005-11
HIGH LEVEL LANGUAGES		GENERAL INFO		012-02
I/O PORTS		BASIC PGM TECHNIQUE	TERM I/O	006-05
IBM/MICROPOLIS FORMATS		HARDWARE		020-12
IMS	INV ANAL SYS	BASIC APPL PGM	DATA BASE	003-06
INKEY ROUTINE*	B.SMITH	8080 PGM TECHNIQUE	TERM I/O	009-04
INTERNAL DATA FORMAT		BASIC DOC	REFERENCE	020-01
INTERRUPTS		HARDWARE		015-05
INVENTORY ONE	BONJOEL	BASIC APPL PGM	BUSINESS	006-09
JUMBLE PUZZLES*	G.RIDING	BASIC APPL PGM	GAME	016-04
KEYBOARD INPUT*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	001-02
KEYBOARD INPUT*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	002-01
LATAH		HARDWARE		017-02
LEFT-FILL WITH ZEROS		BASIC PGM TECHNIQUE		008-07
LIFE		BASIC APPL PGM	GAME	018-05
LIN>BAS	DATASMITH	8080 UTILITY PGM		024-06
LIST	GMS	8080 UTILITY PGM		023-09
MAX-MIN	GMS	8080 UTILITY PGM		023-10
MDOS ALTERATIONS	MICROPOLIS	8080 SYSTEM PGM	OP SYSTEM	008-09
MDOS ALTERATIONS	MICROPOLIS	8080 SYSTEM PGM	OP SYSTEM	012-06
MDOS ALTERATIONS		8080 SYSTEM PGM	OP SYSTEM	014-04
MDOS DISK RECORD ADDR		8080 SYSTEM PGM	OP SYSTEM	009-03
MDOS HANG FIXES		8080 SYSTEM PGM	OP SYSTEM	019-05
MDOS HANG FIXES		8080 SYSTEM PGM	OP SYSTEM	020-12
MDOS ON EXIDY		8080 SYSTEM PGM	OP SYSTEM	021-04
MDOS PDS VERSION 4.0	MICROPOLIS	8080 SYSTEM PGM	OP SYSTEM	014-03
MDOS RELOCATION		8080 SYSTEM PGM	OP SYSTEM	021-08
MDOS UTILITIES	GMS	8080 UTILITY PGM		023-07
MDOS UTILITIES	DATASMITH	8080 UTILITY PGM		024-03
MDOSPATCH		8080 SYSTEM PGM		018-09
MEMORY ALLOCATION		BASIC PGM TECHNIQUE		009-03
MERGE	DATASMITH	8080 UTILITY PGM		024-06
MERGE	DATASMITH	BASIC PGMING AID		020-09
MICOPOLIS SOFTWARE DIR		REFERENCE LISTING	MENTION	004-07
MICRO-LINK		8080 SYSTEM PGM		016-14
MICROPOLIS HARD DISKS	MICROPOLIS	HARDWARE	REFERENCE	003-05

MICROPOLIS USER'S GROUP NEWSLETTER INDEX - CUMULATIVE YEARS 1 & 2

PGMNAME/TOPIC	CO./AUTHOR	PGM/ARTICLE TYPE	CATEGORY	VOL-PG
MICROPOLIS HARD DISKS	MICROPOLIS	HARDWARE		006-02
MICROPOLIS HARDWARE	MICROPOLIS	HARDWARE	REFERENCE	001-02
MICROPOLIS HARDWARE	MICROPOLIS	HARDWARE		013-02
MICROPOLIS HARDWARE	MICROPOLIS	HARDWARE		013-05
MICROPOLIS NEWS	MICROPOLIS	GENERAL INFO		003-06
MICROPOLIS NEWS	MICROPOLIS	GENERAL INFO		006-02
MICROPOLIS PDS SUMMARY	MICROPOLIS	8080 SYSTEM PGM	OP SYSTEM	019-04
MICROPOLIS PREV MAINT		HARDWARE		018-04
MICROPOLIS REPS	MICROPOLIS	REFERENCE LISTING	LIST	003-09
MICROPOLIS S/W		GENERAL INFO	REFERENCE	018-01
MICROPOLIS SOFTWARE		REFERENCE	LISTING	020-14
MICROPOLIS SOFTWARE		REFERENCE LISTING	LISTING	022-08
MICROPOLIS SYSTEM TIPS	MICROPOLIS	GENERAL INFO		009-02
MODEM	D.C.HAYES	HARDWARE		005-10
MODEM PGM	DAVE LOGAN	8080 SYSTEM PGM		006-01
MODEM PGM FOR SOL*	BOB BARNUM	8080 SYSTEM PGM		007-03
MODI-MODII CONVERSION		HARDWARE		002-01
MODI-MODII CONVERSION		HARDWARE		014-05
MODI-MODII CONVERSION		HARDWARE		014-06
MODII-MODI CONVERSION		HARDWARE		013-02
MOTION*	A.PICKERT	BASIC PGM TECHNIQUE		016-02
MTEST	ACROPOLIS	8080 UTILITY PGM		023-04
MUG CONTROL OF MDOS		GENERAL INFO		018-03
MUG DIRECTORY		REFERENCE LISTING	MENTION	006-06
MUG DISK MASTER MENU*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	019-12
MUG LIBRARY		CONTENTS-DISK06	REFERENCE	014-04
MUG LIBRARY		CONTENTS-DISK1001	REFERENCE	020-10
MUG LIBRARY		DISK01 DIR LISTING	REFERENCE	009-01
MUG LIBRARY		DISK01 DIR LISTING	REFERENCE	016-06
MUG LIBRARY		DISK02 DIR LISTING	REFERENCE	011-04
MUG LIBRARY		DISK02 DIR LISTING	REFERENCE	016-07
MUG LIBRARY		DISK03 DIR LISTING	REFERENCE	011-04
MUG LIBRARY		DISK03 DIR LISTING	REFERENCE	016-08
MUG LIBRARY		DISK04 DIR LISTING	REFERENCE	016-08
MUG LIBRARY		DISK05 DIR LISTING	REFERENCE	016-09
MUG LIBRARY		DISK06 DIR LISTING	REFERENCE	016-09
MUG LIBRARY		DISK06 DIR LISTING	REFERENCE	021-10
MUG LIBRARY		DISK07 DIR LISTING	REFERENCE	016-10
MUG LIBRARY		DISK07 DIR LISTING	REFERENCE	021-10
MUG LIBRARY		DISK08 DIR LISTING	REFERENCE	018-11
MUG LIBRARY		DISK09 DIR LISTING	REFERENCE	018-11
MUG LIBRARY		DISK09 DIR LISTING	REFERENCE	021-11
MUG LIBRARY		DISK10 DIR LISTING	REFERENCE	018-12
MUG LIBRARY		DISK11 DIR LISTING	REFERENCE	018-12
MUG LIBRARY		DISK11 DIR LISTING	REFERENCE	021-11
MUG LIBRARY		DISK12 DIR LISTING	REFERENCE	021-12
MUG LIBRARY		DISK18 DIR LISTING	REFERENCE	021-12
MUG LIBRARY		GENERAL INFO		006-01
MUG LIBRARY		GENERAL INFO		009-01
MUG LIBRARY		GENERAL INFO		011-02
MUG LIBRARY		GENERAL INFO	PRICE LIST	011-05
MUG LIBRARY		GENERAL INFO	PRICE LIST	016-05
MUG LIBRARY		GENERAL INFO		021-09
MUG LIBRARY		MODIFICATIONS-DISK06	REFERENCE	016-12
MUG LIBRARY - STDS		GENERAL INFO		020-11
MUG LIBRARY RULES		GENERAL INFO		020-14
MUG MEMBERSHIP DIR		GENERAL INFO		010-04
MUG NEWSLETTER		GENERAL INFO		009-01
MUG NEWSLETTER INDEX		GENERAL INFO	REFERENCE	015-05

MICROPOLIS USER'S GROUP NEWSLETTER INDEX - CUMULATIVE YEARS 1 & 2

PGMNAME/TOPIC	CO./AUTHOR	PGM/ARTICLE TYPE	CATEGORY	VOL-PG
MUG NEWSLETTER INDEX		GENERAL INFO	REFERENCE	021-13
MUG OBJECTIVES		GENERAL INFO		001-01
MUG OBJECTIVES		GENERAL INFO		006-01
MUG OBJECTIVES		GENERAL INFO		007-01
MUG OBJECTIVES		GENERAL INFO		014-07
MUG OBJECTIVES		GENERAL INFO		023-01
MUG SOFTWARE HOUSE		GENERAL INFO		019-01
MUG SUPPLIES		GENERAL INFO	PRICE LIST	024-18
NEVADA COBOL	ELLIS COMP	HIGH LEVEL LANGUAGE	COMPILER	012-03
OSM		8080 SYSTEM PGM	OP SYSTEM	023-14
P/DIM	SYSTEMATION	BASIC UTILITY PGM		010-02
PAS	INV ANAL SYS	BASIC APPL PGM	PROP MGMT	003-06
PAS	INV ANAL SYS	BASIC APPL PGM	PROP MGMT	008-03
PASSWORDS		BASIC PGM TECHNIQUE		018-08
PAYROLL	DATASMITH	BASIC APPL PGM	BUSINESS	006-03
PENSION PGMS		BASIC APPL PGM	BUSINESS	014-06
PGM DOCUMENTATION*	J.SHAPIRO	BASIC PGMING		015-03
PGMING AIDS AVAILABLE	DATASMITH	BASIC PGMING AID	REFERENCE	006-03
PLOADG	GMS	8080 UTILITY PGM		023-12
PMS	CUSTOM ELEC	BASIC APPL PGM	PROP MGMT	003-07
PMS II	INV ANAL SYS	BASIC APPL PGM	PROP MGMT	008-03
POLY TO MDOS ASSM*	M.KONOPICK	8080 UTILITY PGM		022-04
PONY-PICK	BONJOEL	8080 APPL PGM		019-08
PUBLISHED SOFTWARE		GENERAL INFO		013-04
QUIKSORT*	B.RUDOW	BASIC UTILITY PGM	SORT	003-03
R'S AND E'S		BASIC PGM TECHNIQUE		011-02
REACT	BONJOEL	BASIC APPL PGM	BUSINESS	008-03
READING DISK DIR	ED BURKHARDT	BASIC UTILITY PGM	DISK DIR	012-04
READING DISK DIR	B.ZALE	BASIC UTILITY PGM	DISK DIR	014-04
RECOVER	ACROPOLIS	8080 UTILITY PGM		023-05
RES-CONDENSED		8080 SYSTEM PGM	OP SYSTEM	006-06
RESTORE FILES		8080 UTILITY PGM	DISK FILES	018-05
RIGID DISKS		HARDWARE		018-03
ROUNDING*	B.RUDOW	BASIC PGM TECHNIQUE		002-03
ROUNDING*	J.CALLAWAY	BASIC PGM TECHNIQUE		006-05
ROUNDING*	DAVE LAND	BASIC PGM TECHNIQUE		008-01
SAVING BASIC PGMS		BASIC PGM TECHNIQUE		018-15
SAVING RES		8080 SYSTEM PGM	OP SYSTEM	023-13
SAVING VARIABLES		BASIC PGM TECHNIQUE		004-08
SAVING VARIABLES	B.MITCHELL	BASIC PGM TECHNIQUE		008-08
SAVING VARIABLES*	B.RUDOW	BASIC PGM TECHNIQUE		008-08
SAVING VARIABLES*		BASIC UTILITY PGM		010-02
SCREEN DISPLAY TEST*	J.CALLAWAY	BASIC UTILITY PGM	TERM I/O	006-06
SCREEN DISPLAY TEST*	B.RUDOW	BASIC UTILITY PGM	TERM I/O	006-07
SCREEN DUMP		8080 SYSTEM PGM	OP SYSTEM	018-08
SCREEN EDITOR		8080 UTILITY PGM	EDITOR	018-08
SEARCH		8080 SYSTEM PGM		018-09
SGN FUNCTION		BASIC PGM TECHNIQUE		014-05
SIZE(N)*	G.RIDING	BASIC PGM TECHNIQUE	DISK FILES	016-02
SIZES		BASIC DOC	REFERENCE	002-03
SMASH	DATASMITH	8080 UTILITY PGM		024-07
SMASH	DATASMITH	BASIC PGMING AID		020-09
SOFTWARE AVAILABLE		REFERENCE LISTING	UPDATES	022-14
SOFTWARE VENDOR DIR	MICRO-SERVE	REFERENCE LISTING	LIST	005-05
SOFTWARE VENDOR DIR	MICRO-SERVE	REFERENCE LISTING	UPDATES	006-02
SOFTWARE VENDOR DIR	MICRO-SERVE	REFERENCE LISTING	MENTION	007-10
SOFTWARE VENDOR DIR	MICRO-SERVE	REFERENCE LISTING	NOTES	011-03
SOL 80-COLUMN DISPLAY		HARDWARE		018-14
SORCERER BASIC		HIGH LEVEL LANGUAGE	INTERPRET	018-13

MICROPOLIS USER'S GROUP NEWSLETTER INDEX - CUMULATIVE YEARS 1 & 2

PGMNAME/TOPIC	CO./AUTHOR	PGM/ARTICLE TYPE	CATEGORY	VOL-PG
SORCERER S/W		UTILITY PGM		016-15
SORCERER USERS GROUP		HARDWARE		021-09
SORCERER/MDOS PATCHES		HARDWARE		019-12
SORT RETAINING SEQ		BASIC PGM TECHNIQUE	SORT	003-03
SORT/A	SYSTEMATION	8080 UTILITY PGM	SORT	001-01
SORT/A	SYSTEMATION	8080 UTILITY PGM	SORT	003-01
SORT/B	SYSTEMATION	BASIC-80 UTILITY PGM	SORT	010-05
SPELLBINDER		8080 APPL PGM	WORD PROC	021-16
STATISTICS PGMS	MODERN MICRO	BASIC APPL PGM	BUSINESS	013-04
STRING ARRAYS*	B.RUDOW	BASIC PGM TECHNIQUE	STRINGS	019-02
STRING OPERATIONS*	K.FINDLAY	BASIC PGM TECHNIQUE	STRINGS	016-13
SUBROUTINE LIBRARY		BASIC PGM TECHNIQUE		010-01
SYSLIST	DATASMITH	8080 UTILITY PGM		024-07
SYSTEM MEMORY		8080 PGM TECHNIQUE		017-03
SYSTEMATION DISCOUNTS	SYSTEMATION	GENERAL INFO		009-01
TABLE HANDLING		BASIC PGM TECHNIQUE		014-02
TABLE HANDLING*	J.HARDEN	BASIC PGM TECHNIQUE		014-07
TABLE HANDLING*	N.DEMBINSKI	BASIC PGM TECHNIQUE		015-01
TAPEREC		BASIC APPL PGM	MUSIC	011-06
TAX PGMS	SYNTAX CORP.	BASIC APPL PGM	BUSINESS	006-04
TAXPRO		8080 APPL PGM	BUSINESS	017-04
TEXTWRITER		8080 APPL PGM	WORD PROC	017-05
TICTACTOE	MUG LIBR	BASIC APPL PGM	GAME	008-06
TOKENIZE	ACROPOLIS	8080 UTILITY PGM		023-06
TRACK DENSITIES		8080 SYSTEM PGM	OP SYSTEM	017-04
TX*	B.RUDOW	BASIC UTILITY PGM	DISK FILES	004-01
UNPROTECT	SYSTEMATION	CP/M UTILITY PGM		009-05
UTILITY PGMS AVAILABLE	BONJOEL	GENERAL INFO		006-07
UTILITY PGMS AVAILABLE	SYSTEMATION	UTILITY PGM	REFERENCE	007-09
UTL-1 PGM PACKAGE	SYSTEMATION	8080 UTILITY PGM	DISK FILES	005-12
VARIABLE ALLOCATION*	B.SMITH	BASIC PGM TECHNIQUE		008-06
VARLIST	DATASMITH	8080 UTILITY PGM		024-03
VECTOR GRAPHIC SYSTEM	VEC GRAPH	HARDWARE		003-05
VERIFY	ACROPOLIS	8080 UTILITY PGM		023-06
VERSATILE CDS		HARDWARE		017-14
VIEW	ACROPOLIS	8080 UTILITY PGM		023-03
WAMSORT	BONJOEL	8080 UTILITY PGM	SORT	006-09
XFILES	GMS	8080 UTILITY PGM		023-07
XREF	SYSTEMATION	BASIC PGMING AID		004-10
XTYPE	GMS	8080 UTILITY PGM		023-09
YES/NO INPUT RESPONSES*	B.SMITH	BASIC PGM TECHNIQUE		009-04
YES/NO INPUT RESPONSES*	ED BURKHARDT	BASIC PGM TECHNIQUE		012-05
Z80 ASSEMBLER		8080 PGMING AID	ASSEMBLER	018-10

Reference Manual, Revision A, Feb. 1, 1980, page 1-2, states that MEMORITE uses an incompatible operating system.

If you try to read a MEMORITE file into SCOPE, the system returns the message:

BAD TEXT FILE

Appending a CRLF to each line of a MEMORITE file is all that is necessary to make them accessible to SCOPE or useful in SENDing by telecommunications. A simple ED instruction does this, but this is a nuisance. There is a direct way to achieve the same result, not documented in my Vector MEMORITE III PRIMER Version 1.1, Revision A, Feb. 1, 1981.

As is revealed in the VECTOR CONECT Users Manual, it is possible to create a "raw" file while in MEMORITE. Although the instruction is not explicitly given there, the consistent use of mnemonics by Vector suggested writing a document in MEMORITE using:

WD R

for

>.cWD <R>aw

This worked and is convenient for the user. I often generate MEMORITE files which I want to process as both standard MEMORITE files and under other programs as ASCII files; by writing a MEMORITE version (CLASSM) and a "raw" version (CLASSR), I can have it both ways.

.....

SPECIAL VECTOR SALE

A Vector Graphic dealer here in Huntsville is having a special sale which will be of interest to any of you contemplating an update of your current Micropolis-based system.

Model	Retail	Sale
Vector 2600	3,995	2,996
Vector 3005	5,495	4,121
Vector 5005	7,990	5,993
5005 Terminals	1,650	1,250

All the Vectors use a double-sided drive which can read your MOD II or MOD IV disks. The 2600 has two external floppy drives, with 64K, 5 S-100 slots and a memory mapped video built into a terminal. The 3005 substitutes a 5Mb hard disk for one floppy. The 5005 is a multi-user (maximum of 5) system. It comes with one terminal.

If you're interested, contact Bill Hill at Technical Data Systems, 205/882-1300, or write him at 2227 Drake Ave SW, Huntsville AL 35805.

It's amazing what is going on in the microcomputer world. It was just last March that I bought my Black Hawk, which is essentially a Vector 3005, for $5,500 - wholesale.

Vector has introduced the Vector 4. It is an 8-bit/16-bit system with color capabilities. Prices are similar to the list prices above. I guess that's why they're having a sale on the "old" equipment.

.....

****

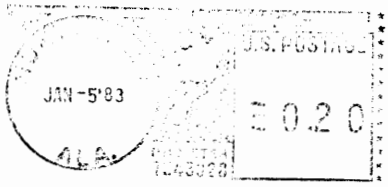
Published Monthly by the MUG
Subscription rates:
U.S., Canada, Mexico; $18/year: Other, $25/year

FIRST CLASS MAIL

FIRST CLASS MAIL

MICROPOLIS USERS GROUP

Buzz Rudow, Editor
604 Springwood Circle
Huntsville AL 35803
(205) 881-1697



FIRST CLASS MAIL