MICROPOLIS USERS GROUP

MUG Newsletter #25 - August 1982

****

A COMPARISON OF MICROPOLIS AND MICROSOFT BASIC

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

Probably the most widely used BASIC in the world is from Microsoft. Designed for use on 8080 or Z-80 microcomputers, Microsoft Basic is available for several different operating systems, and is extremely popular even on the Apple, where it requires its own processor. Microsoft is not available to run under MDOS, but if you use CP/M it is probably the language you will see most often. This article provides a comparison between Microsoft Basic and Micropolis Basic, written from the perspective of someone who is more familiar with Micropolis. I have tried to be fair in the comparison, but I do have my own opinions, which are identified as such.

Since both languages are versions of Basic, most of the commands and functions are the same, as you would expect. I have found it relatively easy to convert programs from one language to the other once I learned the implications of the differences between them. However, the differences are important, making some types of program structures difficult or impossible to translate.

ENTERING, LOADING, AND SAVING PROGRAMS

Entering a program is quite similar in both languages. BASIC is started by typing BASIC from the operating system level, which loads the interpreter and executes it. Basic statements are entered in exactly the same way, and both languages have identical program edit facilities. Microsoft, however, has an AUTO command which generates line numbers automatically. This is a useful timesaver when entering a new program. If an automatically generated line number already exists in the program, Microsoft warns you by printing an asterisk after the line number, preventing accidental overwriting of existing program lines. A very nice feature of Microsoft Basic is that it allows long variable names instead of the simple letters used in the minimal configuration provided by Micropolis. In both languages a variable terminated by a % sign is an integer and a variable terminated by a $ is a string. Microsoft allows this to be overriden, however, by declaring variables real, integer, or string.

Both Basics save programs on disk with a SAVE command, but Microsoft does not differentiate between old files and new files. If the file being saved does not exist on disk, Microsoft creates it. If the file does exist on disk, Microsoft replaces it. This is in contrast with Micropolis, which requires that you specify whether the file is new or not with the N: prefix. The Microsoft scheme is convenient but more dangerous. If you accidentally give different programs the same name, the previously existing program will be destroyed. Microsoft also has three different formats for saved files.

Ordinarily, a program is saved in a binary form with all keywords converted to tokens like Micropolis does. However, a program can also be saved as ASCII text or in a "protected" format. The ASCII text format is very useful because it allows the use of a text editor on the program or transmission through a modem. ASCII programs can be loaded and executed by Basic, but the loading process takes longer because each line must be tokenized before it is stored in memory. The protected format is similar to the regular binary format, except each byte is somehow encrypted before it is written to disk, so that the file looks like so much garbage except to Basic. The first byte in the file tells Basic whether or not it is protected so that when it is loaded, it may be decoded back to an executable form. Once a program file is stored in a protected format, it may not be listed, changed, or unprotected, even by the original programmer. In theory, this feature should protect your software from being stolen by unscrupulous third parties, but it actually only provides protection against the ignorant. Once a protected program is loaded, its memory image is the same as an unprotected program, so the debugger can be used to do a little fiddling with the protect/unprotect byte and the memory image can be saved on disk in an unprotected format. I found this much easier than trying to figure out what the encryption scheme was.

PROGRAMMING STRUCTURE

The way the FOR - NEXT loops work in the languages is different. In Micropolis, a FOR-NEXT loop is always executed once even if the ending condition is exceeded at the outset, and the loop always exits with the loop variable equal to the TO value (last value used). This is nonstandard. The standard way, and the one Microsoft uses, is that a FOR-NEXT loop will not be executed at all if the end condition is exceeded at the start. For example, FOR X=9 TO 5 ..... NEXT X won't do anything in Microsoft. Microsoft always exits the loop with the looping variable exceeding the TO value (first value not used). It is important to remember this distinction. Microsoft also has a WHILE - WEND loop, nice, but it can be duplicated in Micropolis with the same number of statements. The fact that Microsoft has an ELSE clause associated with an IF - THEN statement is a definite advantage, however.

NUMBER CRUNCHING

Both versions of Basic have the same or very similar math operators and functions, so they will both perform the same types of mathematical operations. The few differences that exist are that Microsoft has an XOR function and a SWAP function (useful in sorts), but is missing MIN, MAX, LOG (common), and FRAC, which are included by Micropolis.

The big difference is in the precision and the way math is performed internally. Micropolis has variable precision that varies anywhere from 4 to 53 significant figures, as set by the SIZES statement. This allows scientific or engineering computations about as accurate as you would ever need them, sufficient for interstellar navigation if you care to install Micropolis Basic on your starship. Variable precision is a unique feature, and gives little Micropolis based micros a computational accuracy that beats many high level languages on big mainframe computers.

Microsoft Basic gives you two choices of precision. Single precision, which is 6 digits accuracy, and double precision, which is "about" 16 digits accuracy. Both single precision and double precision variables may be mixed in the program. A double precision variable is identified by terminating it with a # sign or by declaring it as double precision. Integer variables in Microsoft are strictly 15 bit numbers with a sign, limiting them to + or - 32767.

The languages differ considerably in internal storage format. Micropolis uses BCD (Binary Coded Decimal) where decimal digits are packed two to a byte and math is carried out much like humans do it, in decimal. Microsoft, however, uses pure binary storage and arithmetic, which means that numbers entered in decimal from the keyboard must be converted to binary for storage and values to be printed must be converted from binary to decimal on the way out. Since all the math is carried out in pure binary, Microsoft introduces rounding errors with double precision that are extremely annoying. For example, dividing .1 by 10 in Microsoft's double precision does not result in .01, but some number that is about 150 trillionths more, much

less accuracy than the 16 digits claimed. This means that, unless the programmer adds some kind of round-off routine, balance sheets don't always balance exactly, and numbers that should be zero don't always compare equal to zero. The binary math means that Microsoft is generally a faster number cruncher than Micropolis, because the variable precision and BCD arithmetic slow up the latter. BCD also requires more storage. If I were programming for accuracy, however, I'd wait the few extra milliseconds for the right answer.

STRINGS

All the common string functions are present in both languages, but there are a few differences. The REPEAT$ function in Micropolis is called STRING$ in Microsoft, but Microsoft only repeats a single character, rather than a whole string as is done by Micropolis. Both languages have a function to find substrings (INDEX vs INSTR), but Microsoft has no counterpart of the Micropolis VERIFY. Also missing from Microsoft is the extremely useful FMT function. In its place, PRINT USING may be used for formatting output, but there is no general purpose formatting function that can be assigned to a string. Included by Microsoft, but not by Micropolis, are HEX$ and OCT$ functions which convert numbers to strings in hexadecimal or octal notation. This is extremely handy for utilities, memory dumps, etc. String lengths in Microsoft are dynamically allocated, so it is not necessary to dimension strings because Basic will automatically provide space for up to 250 characters per string. Micropolis, on the other hand, has fixed length strings and forces you to specify string lengths with the SIZES or DIM statements.

Microsoft's dynamic string allocation is a two edged sword. On one hand, it makes the programmer's job much easier by giving you as much string length as you need up to the 250 character maximum. On the other hand, the internal scheme for string allocation causes serious performance problems in large programs that do a lot of string handling. Every time a string is assigned (or reassigned) a value, the old string storage for that variable in memory is abandoned and the new value for the string is stored in previously unused memory elsewhere. Actually, this is a fairly fast operation since all that is required is for an internal pointer to be changed, but the memory used by the old contents of the string is not recovered. The result is that Basic gobbles up a little more memory each time a new value is assigned to a string. Eventually, when there is less than 256 bytes of memory left, all that abandoned memory must be recovered with a procedure called "garbage collection" (really). This is a complete reorganization of variables in memory and can take longer than 30 seconds to complete! Long programs that don't have much memory left over for variables need garbage collection more often and suffer by being much slower than short programs.

The last big difference between the languages are the disk storage functions. Micropolis has one kind of file, random access, and Basic can only write ASCII characters (usually) to the file. However, there are a large variety of disk related functions in Micropolis Basic that make using disk storage very easy and efficient as far as coding is concerned. Programs can be made very user friendly and error tolerant, because disk errors can be trapped and handled with software, and the status of the disk drives and files can be tested, avoiding disk errors through information. The Micropolis file system has the disadvantage of being of fixed record length. This means that, unless you provide your own blocking and deblocking logic, short records waste a lot of disk space and records longer than 250 characters are not possible.

Microsoft has two kinds of disk files. Sequential (more properly called "stream") files, and Random files. Sequential files have no record structure at all and are accessed with PRINT and INPUT statements. A Sequential file is composed of data separated by commas, carriage returns, and linefeeds, just as if it were being entered from the keyboard, and is read and written accordingly. With a Sequential file, you can write blocks of data in varying lengths and in any format you want, but you have to make sure you read it back in the exact way it was written. There is no way to extend a sequential file without rewriting it completely, so they are not suitable for large files that must be added to or updated frequently. Opening a Sequential file for output will either replace any file with the same name, or create a file if the name isn't in the directory, so you have to be careful to first open the file for input to see if it exists if you are concerned about destroying data.

Microsoft Random files have the advantage of being opened with any record length you desire, with Basic providing all blocking and deblocking functions without regard to the disk's physical sector size. Fields within a record are generally of fixed length, with a number of record packing and unpacking functions provided for this purpose. Numeric data in a Random file is written to disk in binary format which uses less space, so this feature, coupled with user-defined record length allows much more efficient storage of data than Micropolis provides. The price you pay to use random files to advantage with Microsoft may require as much as ten times more code to implement than with Micropolis, and much more careful planning of record layout. Microsoft Basic is not tolerant of disk errors and there is no way to trap drive not up or permanent I/O errors. When these kind of disk errors occur, it invariably results in a crash, sending you back to the operating system with no opportunity to close files or otherwise provide an orderly retreat. This makes it difficult, if not impossible, to write an error-tolerant program in Microsoft.

While both languages have facilities for calling and passing parameters to assembly language subroutines, Microsoft (at least version 5.2) does not have a command to load such a subroutine! We assume that the user has to use the debugger or some other facility to load this program before getting Basic up. I note , however, that the IBM Personal Computer Basic (which is also Microsoft) has fixed this with binary load and save commands.

SUMMARY

Which one would I choose? If you aren't using Micropolis drives there isn't much choice. If you are, it depends on what you want to do and your programming style. I prefer Microsoft's long variable names generally faster numeric processing if accuracy isn't too important. For accuracy and error handling ability Micropolis is hard to beat. I am particularly fond of the FMT function, which I use for all sorts of general purpose numreric formatting. It is also much easier to use than PRINT USING, even in print statements. If I had to make a choice, I'd say that Micropolis Basic is the more powerful language, all things considered.

A Basic Cross Reference Listing is shown, starting on page 7, following Zot's assembly listing.

.....

BUILDING THE CHEAP COMPUTER, PART IV

By Zot Trebor

If you've followed me through the previous articles, you'll appreciate that the major problem is not the simple replacement of one monitor routine with another. If I had a working PROM programmer this whole idea of stuffing a video driver into our RESident module would never have come up. But I don't. So it did.

The open space left in the RES is akin to a natural resource for we MUGgers. I hate to waste it on something that properly should be frozen in a PROM. If I get a PROM programmer working, expect to see

me telling about it in the Newsletter; perhaps we can come up with a Cheap Computer Standard Monitor or something equally silly.

This article will describe how to install the video driver within the RES without crashing the system, something I did countless times until I got the hang of it.

The source program is named "VIDEO" and is reasonably well documented via imbedded comments. The program is assembled in the usual manner except its object code is offset in memory. This is necessary because the MDOS does not allow loading a program, by either the LOAD or MOVE commands, into an area of memory occupied by the operating system.

ASSM "1:VIDEO" "1:VID.OBJ" "PTS"

will generate the necessary documentation and make a master history or backup file of the program. A further assembly,

ASSM "1:VIDEO "" "M" "8000"

will produce the offset program. The assembly should immediately be followed with a

SAVE "1:VID.OBJ1" 861B 8740 4

to preserve the offset program.

With both MDOS and RES in memory, the offset program should be loaded as...LOAD "1:VID.OBJ1". Now all the necessary parts of the conversion are in memory.

It is now necessary to reset the system and return to your primitive monitor, as MDOS will not allow the direct overlay of the RES area. Using your monitor, move the new program from its offset area into its proper area as in the form....

M861B 86B7 061B<cr> (using the SSM monitor form)

M8711 8735 0711<cr>

Notice that the driver is loaded in two parts. The console routines in RES, which are being replaced by the driver, run from 61B to 6CA, at which point the printer drivers begin. Rather than re-assemble the printer drivers, I've simply loaded around them, starting the second portion of the video routines at 711. This is more fully explained in the assembler listing.

Now the new video driver is in place but is not connected to the rest of the system. Connection is made by plugging the new subroutine addresses for CDIN, CDOUT, CDBRK and CDINIT into the lookup table beginning at 04F8. CDIN does not need a new address; the address stays the same as it is the first subroutine.

The new addresses are:

04FB = CDOUT = 064E

04FA = CDBRK = 062C

04FC = CDINIT = 0637

If you enter these using the SSM monitor, your input should look like this:

S4F8 3B-4E 06- 54-2C 06- 6B-37 <cr>

Notice that you don't have to enter anything if you are not changing that locations contents. Pressing the space bar advances the address. In the case above we changed only the low-order positions of three addresses.

Now the new video driver has been inserted into memory and the new addresses have been entered in the console table. Again, using the SSM monitor, go to the warm-start address and let's see if it flies. Enter G4E7<cr> and you should get an immediate screen clear and the printing of the MDOS legend on the screen. If you didn't, it's back to the drawing board.

But we aren't done yet. Micropolis doesn't allow distructive backspaces, they like to put slashes and stuff on the screen. We've taken care of that, but we must adjust our character count; each time we backspace, we are reducing the character count.

I wish I had some neat little routine to take care of this, but I don't. What I've done is to put a patch on the MDOS and it is very crude.

Backspaces are handled by MDOS in a routine called DEV010. It's in Appendix E, page 7 of your manual. Memory address is 599 thru 5A0. Here is what I did. I put an unconditional jump at location 59E to take us down to a clear area at 5E4. At 5E4 I put a little routine to decrement the H-register, ie, reduce the character count for the line. In the next routine, the character count is compared to line length, so everything works out.

To install this patch, make the following entries under MDOS

ENTR 59E

C3 E4 05/<cr>

Which takes us down to the patch...

ENTR 5E4

0E 5F 25 C3 A1 05 / <cr>

which does the work and takes us back to the next routine, DEV020, where the line length gets checked.

Now we've done it. We have a neat little video driver nestled down in our RES module. To save it, just save RES. See para. 2.2.6, page 2-33 of your Micropolis manual to save RES.

We now have a video driver tucked away in RES, always available and certainly much more useful than the one in the SSM monitor. But should we stop there? I don't think we should. For one thing, the video is still addressed at B000 and that takes a devil of a chunk out of our memory space. I'd like to tuck the video away up at high memory, say about F000. The only problem here is the fact that the SSM monitor is also at F000. Hummmm.

If we look at the SSM cpu card we see that we can shut off the PROM's that hold the monitor...and if we no longer need the monitor; the video driver now being in RES, why not try it? This leads to the problem of how to bring up the system if we don't have a monitor. Again, looking at the SSM cpu card we see that we can address the vector jump on reset to anywhere, and that includes to the start of the Micropolis boot-strap routine. If we want to keep our high memory usage compact it would be nice to strap the Micropolis to, say E800 where it can tuck itself in under our new video location of F000. Lets see how that works.

Re-strapping the Micropolis is done by physically unsoldering and removing one jumper on the Micropolis board. The instructions for doing this are covered in para. 2.1.4.1, page 2-4 of the manual. Pages 2-5 and 2-6 are full-page illustrations that make the job virtually goof-proof. So re-strap the board (the little jumpers are called 'straps': old-time computerese to confuse the novice) and try it out. Don't change anything else until you are satisfied that your re-strapping works okay.

With the controller board working at E800 we now shut off the PROM's on the SSM cpu card by pushing the right-hand side of switch S2-P. Thats the third switch position up from the bottom of switch S2. The PROM's are now turned off. The monitor is dead. Long Live the RES module!

Now we need to get the reset to take us to the Micropolis bootstrap at location E800. Switch S2 is also used to set the address. E8 is 1110 1000 but we are only setting the high-order five bits

(1110 1---). The high-order position is the top switch section on S2 and a switch section is a '1' when the left-hand side of the switch is down. Use a pencil or something similiar to set the sections to 11101. Now try it. Press your reset button and latch down the disk at the same time. You should hear the familiar click and in a moment the legend should appear on the cleared screen. It's almost like having a real computer!

Now lets look at the problem of moving our video display up to high memory. The video board is set to locaction B000 as required by the SSM monitor...which we are no longer using. We can put it anywhere we want now, so long as the video driver in the RES module knows where it is. The problem here is that once we re-address the video we can no longer see what the devil is going on. Risky. Better to make any changes to the program first and then reset the video board address.

The new video driver is designed to accept new video start and end addresses dynamically so it is no problem to get in there and set up for a different location. Look at the assembly listing, line numbers 670 thru 900. The constants shown in lines 690 and 760 allow the video initialization subroutine (CDINIT) to set-up the starting and ending addresses of the video screen, as well as the starting position of the cursor. If we were changing the start of the screen from a program, for example, to partition the screen to preserve a heading or operator instructions, we would poke the new value into the buffer we have assigned (SCRNEND). Each time the system was initialized, the standard start and end addresses would be used. Neat. But we want to actually change the standard addresses and that means that once we do, we had better change the video board switch settings to match those addresses or else the 'video' will be down in RAM somewhere, displaying its heart out to a neighboring resister. Here is how we do it.

Using the MDOS routine for ENTR'ing data, (remember, the SSM monitor has died and gone to Corona).

```
ENTR 63A

F0 F4 /<cr>

ENTR 641

F3 /<cr>
```

You just did it. The next time you initialize the system, the video will be addressed at F000 to F3FF. It doesn't do it now because you haven't changed the screen locations in the buffers.

Now reset the address of the video board. This one is a snap because the difference between B000 (1011) and F000 (1111) is so obvious; just push the second switch element into alignment with the other three. Bingo! The screen just went dead. Now don't touch anything for a minute. It wouldn't show up on the screen if you did. Being super-careful, type in EXEC 4E7<cr>. Micropolis will warm-start and re-initialize the video to the new addresses.

Now you have a different RES module than that which is on your disk. What's more, if you were to hit the reset button at this point you would get nothing because your video board is now at a different location than the video driver in the RES module. Be Careful! We need to copy the RES module to disk. Follow the same instructions as before.

Complicated, isn't it? With the new RES safely backed up on at least two disks, give it the acid test; shut down the system completely. Now power back up, depress the disk latch and hit reset. If it dosen't work, take two aspirin and call Buzz in the morning.

.....

LETTERS

DOUBLE-SIDED DRIVES

Buzz,
I've just recently acquired the 1053-M4 dual drive unit (double sided drive). Micropolis informs me that currently no vendor supplies CP/M configured with double-sided disk drives in its BIOS.

With the help of some friends in my SORCERER's user group, I hope to modify a single-sided BIOS to drive these new drives. However, I am having some difficulty in getting technical information/methodology for driving the controller. Micropolis has sent me some information, but not really enough.

Does perhaps someone in your organization have some more information on this subject?

Also, since EXIDY is no longer in business, I'm having difficulty in getting the 'patches' necessary to bring up MDOS & MICROPOLIS BASIC on my machine.

I'm hoping that someone else in your group has also been this route, and might be able to help.

The versions of software supplied on the diskettes are all V4.1. I am confused about the need to apply the V4.0 patches to the RES & MDOS, as the enclosed documentation has suggested (SIB's #12 thru #16). Any suggestions of yours would be most appreciated.

And finally, my eventual desire is to get CP/M. However, I don't want to start to acquire a large set of files, until I am certain they can be converted into CP/M formatted files. If anyone has done these things, I would appreciate hearing of their experiences.

Jonathan Burnett, 904/358-1480
5422 Missouri Ave., Jacksonville FL 32205.

Jonathan:
I've been asking around about a CP/M for the MOD IV, but haven't found one yet. There supposedly is one. I'll keep looking.

Enclosed is an Exidy patch from MUGie Bruce Taylor which may help with the Micropolis system. If you are having further problems, give me some specific examples. We'll see what we can do.

My guess is that you should put in all the Version 4.0 patches. Though I've never seen Version 4.1, it must be compatable with 4.0. I would think that the major differences are in the disk access routines. All entry points should be the same as 4.0. I've never heard of software being incompatible. Since the patches have to do with BASIC strings and stuff. I think the code is identical.

Yes, you can convert MDOS files to CP/M files. You will find an article on this in newsletter #19, page 6. Another option is to use BASIC/S while you are on MDOS, and then use BASIC/Z when you go to CP/M. I think these BASICs are super. Actually, they are both called BASIC/Z, now. Nothing like them in the CP/M world.

By-the-way, you might get some help from Dynasty, 14240 Midway Rd., Dallas TX 75234, (214) 386-8634. I hear that they have taken over the Exidy products, though I don't know if there is a Sorcerer anymore. I assume you know of The Sorcerer's Apprentice. See newsletter #21, page 9.

If any of the MUG members have gotten CP/M running on the MOD IV, please let both Jonathan and me know about it.

.....

Title: VIDEO

```
0010 ************************************************************
0020 *
0030 *  TITLE: VIDEO
0040 *  This is a video driver program written to fit within
0050 *  the Micropolis RESident module.
0060 *  Written by Zot Trebor, Dec. 1980.
0070 *
0080 ************************************************************
0090 *
0100 *   First, an explaination:  The system is brought up under
0110 * control of the SSM monitor which requires the video to be
0120 * addressed at B000 and the keyboard at ports 2 (status)
0130 * and 3 (data).  Initial values in this program are to
0140 * satisfy the SSM monitor.  Additional instructions have
0150 * been provided via the Micropolis Users Group for the
0160 * actual installation of this driver.
0170 *
0180 *  A few EQUates for information
0190 KBStatus EQU  02H      ; The keyboard status port
0200 KBData EQU   03H      ; The keyboard data port
0202 BellOut EQU  0FEH     ; The 'Bell' Outport
0210 *
0220 *
0230        ORG   061BH    ; Start at the normal CDIN address
0240 *
0250 ************************************************************
0260 *    CONSOLE DEVICE INPUT ROUTINE
0270 ************************************************************
0280 *
0290 CDIN   IN    KBStatus ; Get Key Board status
0300        ANI   1        ; .AND. with the flag mask...
0305                       ; This isolates the low-order bit
0310                       ; ..I use InPort 2 for several types of status
0320                       ; ..bit 1 is for the KB.
0330        XRI   0        ; .OR. with status mask
0340                       ; This determines if it is a 1 or zero
0350        JZ    IN010    ; Is it a zero?
0360        JNZ   CDIN     ; No, loop until it is
0370        NOP            ; Placeholder
0380 *
0390 IN010  IN    KBData   ; Get key board data
0400        MOV   B,A      ; All Micropolis software expects
0410                       ; the character in the B-reg
0420        RET
0430 *
0440 ************************************************************
0450 *   CONSOLE DEVICE BREAK CHECK ROUTINE
0460 ************************************************************
0470 *
0480 CDBRK  IN    KBStatus ; Get Key board status
0490        ANI   1        ; Isolate LOBt
0500        XRI   0        ; Is it a 1?
0510        RNZ            ; RETurn if it's a 0
0520 *
0530 CDBRK0 IN    KBData   ; It was a 1, get the data
0540        MOV   B,A      ; Put into B for Micropolis
0550        RET
0560 *
```


Title: VIDEO

```
0570 ************************************************************
0580 *   CONSOLE DEVICE INITIALIZATION
0590 ************************************************************
0600 *                  NOTE
0610 *  A memory-mapped video device does not require initial-
0620 * ization as would an I/O-mapped device which must comm-
0630 * unicate with the computer via USART.  Initialization
0640 * in this case is merely setting the initial screen para-
0650 * meters.
0660 *
0670 CDINIT PUSH   H        ; We are going to destroy these
0680        PUSH   B        ; registers, so save contents
0690        LXI    H,0B0B4H ; Get screen address hi-order
0700                        ; bytes, B0 & B4.  These are
0710                        ; for the SSM monitor and will be changed
0720                        ; when the program is installed in RES.
0730        SHLD   SCRNEND  ; Store the hi-bytes in SCRNHOME
0740                        ; and SCRNEND.  They are adjacent so we
0750                        ; need only address SCRNEND
0760        LXI    H,0B3FFH ; Get the last screen address for
0770                        ; screen-clearing
0780        SHLD   CURSPOT  ; Store it in the the Cursor
0790                        ; Position buffer
0800        MVI    B,0CH    ; Get the Screen Clear code..
0810        CALL   CDOUT    ; ..and clear the screen
0820        POP    B
0830        POP    H        ; Restore the registers...
0840        XRA    A        ; ..and turn off the Carry Flag (CY)
0850                        ; to tell Micropolis we're done.
0860        RET             ; Done with initializing the screen.
0870                        ; We will start with the screen clear
0880                        ; and the Micropolis legend in the upper-
0890                        ; left corner
0900 *
0910 ************************************************************
0920 *  CONSOLE DEVICE OUTPUT ROUTINE (VIDEO)
0930 ************************************************************
0940 *
0950 CDOUT  PUSH   PSW      ; Save ALL registers
0960        PUSH   B
0970        PUSH   D
0980        PUSH   H
0990        MOV    A,B      ; Get the character into A-reg
1000        LHLD   CURSPOT  ; Get cursor address into HL
1010 *
1020 CNTRLX EQU    18H      ; Clears current line
1030        CPI    CNTRLX   ; is A = 18H?
1040        JZ     CLEARLINE ; Yes, go clear the line
1050 *
1060 *  Read the following carefully
1070 *
1080        LXI    D,SCROLL ; Get the Scroll Routine address...
1090        PUSH   D        ; ..and stuff it on the stack (!)
1100                        ; Local RETurn's will now fall through SCROLL
1110 *
1120        MVI    M,20H    ; Get a space character and erase
1130                        ; the cursor.  Fall thru...
1140 *
```

```
1150 RBOUT  EQU     5FH         ; Micropolis has chosen to change the
1160                            ; normal BackSpace (08) into 5F...
1170                            ; I have no idea as to why.
1180                            ; So, 5F is a BS and must be treated seperately.
1190        CPI     RBOUT       ; Is A = 5F?
1200        JZ      RUBBIT      ; Yes, go rub it out.
1210        CPI     20H         ; No.  Is A a printing char?
1220                            ; ie, >20H?
1230        JNC     PAINT       ; Yes, A was < 20H so go
1240                            ; to printing character routines
1250 *
1260 * At this point we know the character is not a line erase, not
1270 * a backspace and not a printable character, therefore it must
1280 * be a control character.
1290 *
1300 CR     EQU     0DH         ; Carriage Return Code
1310 *
1320        SUI     CR          ; Subtract 0D from the A-reg
1330        JZ      CARTN       ; If equal, go do a carriage retrn
1340 *
1350 FF     EQU     0CH         ; Form Feed Code
1360                            ; FF will clear the screen and
1370                            ; home the cursor
1380        INR     A           ; Follow the logic here...
1390        JZ      FFRTN       ; Go do a form feed
1400 *
1410 VT     EQU     0BH         ; Vertical Tab Code
1420                            ; We get this for free, so
1430                            ; put it in
1440        INR     A           ; If A = 0, char is 0B
1442        JZ      VERTAB      ; Homes the cursor
1450 *
1460 *   No other useful control codes.  If you have a special
1470 *   application, include the codes here or before the RET
1472 *
1473 *  Don't assemble the following unless you have space
1480 *
1490 *BELL EQU 07H ; This one is just for fun
1500 * INR A ; A = 0A
1510 * INR A ; A = 09
1520 * INR A ; A = 08
1530 * INR A ; A = 07
1540 * JZ Bell ; If it's a 7, go to Bell
1550        RET             ; We will fall through SCROLL
1560 *
1570 ************************************************************
1580 *  PAINT - PUTS PRINTING CHARACTERS ON SCREEN
1590 ************************************************************
1600 *
1610 PAINT  MOV     M,A         ; Put the character into screen memory
1620        INX     H           ; Advance the pointer
1630        RET                 ; We will fall through SCROLL
1640 *
1650 ************************************************************
1660 *   SCROLL ROUTINE (NORMAL EXIT)
1670 ************************************************************
1680 *
1690 SCROLL LDA     SCRNEND     ; Get Screen End hi-order byte
```



```
1700        CMP     H           ; Have we run off the screen?
1710        JNZ     CURP        ; No, go restore the cursor
1720 *
1730        LHLD    SCRNEND     ; Get the two-byte hi-order bytes
1740                            ; showing screen home and screen end
1750                            ; H=screen home, L= screen end
1760        MOV     A,H         ; Put start into A
1770        SUB     L           ; Follow the logic here:
1780                            ; Start = B0, End = B4
1790                            ; Subtract B4 from B0 = FB, a negative
1800                            ; number
1810        MOV     D,H         ; Put the hi-byte into D-reg...
1820        MVI     L,40H       ;.. and line length into L
1830                            ; HL now represents the start of
1840                            ; the 2nd line on the screen (B040)
1850        MVI     E,0         ; Set E to zero...
1860        MOV     C,L         ; Get 40H into C
1880        MOV     B,A         ; And put the result of our sub-
1890                            ; traction into C
1900 *
1910 *  At this point the registers contain the following:
1920 *    HL = B040, start of 2nd line
1930 *    DE = B000, start of 1st line
1940 *    BC = FB40, BC will equal 0 when we have moved the
1950 *                characters for 15 lines, ie, line 2 will move
1960 *                to line one, line 3 to line 2, etc.  The 16th
1970 *                line is handled seperately by CLEARLINE
1980 *
1990        CALL    REVOM       ; Call the backwards move routine
2000        DCX     H           ; Adjust the HL register
2010                            ;...and fall thru to CLEARLINE
2020 *
2030 ************************************************************
2040 *    ERASE THE CURRENT LINE
2050 ************************************************************
2060 *                 NOTE
2070 *  Erases the current line if a Control-X.  Erases the 16th
2080 *  line if fallen into from SCROLL
2090 *
2100 CLEARLINE MVI A,3FH    ; Get the line length..
2110        MOV     D,A         ; ... into D-reg
2120        ORA     L           ; Find logical end of current line
2130        MOV     L,A         ; Set HL to logical end of line
2140 ERASE  MVI     M,20H       ; Write a space to the location
2150        DCX     H           ; Backup HL to next position
2160        DCR     D           ; Backup the line length counter
2170        JNZ     ERASE       ; Keep erasing til D=0
2180                            ;...then fall thru to CURSOR RESTORE
2190 *
2200 ************************************************************
2210 *     CURSOR RESTORE ROUTINE
2220 ************************************************************
2230 *
2240 CURP   LDA     CURSCHAR    ; Get the cursor character
2250        MOV     M,A         ; Write it to the screen
2260        SHLD    CURSPOT     ; Save the location for the next
2270                            ; character...
2280                            ;...and fall thru to the normal exit
```

```
Title:  VIDEO

2290 *
2300 ************************************************************
2310 *    RESTORE REG'S & RETURN
2320 ************************************************************
2330 *
2340 EXITER POP     H         ; Restore all registers
2350        POP     D
2360        POP     B
2370        POP     PSW
2380        XRA     A         ; Turn off the carry flag as signal..
2390        RET               ;...to Micropolis
2400 *
2410 *
2420 ************************************************************
2430 *       BACKSPACE ROUTINE
2440 ************************************************************
2450 *
2460 RUBBIT DCX     H         ; Backup the screen pointer
2470        RET               ; ...Return via SCROLL & CURP
2480 *
2490 ************************************************************
2500 *        CARRIAGE RETURN ROUTINE
2510 ************************************************************
2520 *
2530 CARTN  LXI     B,40H     ; Get line length + 1..
2540        MOV     A,L       ; Get cursor positon lo-byte into A
2550        ANI     0C0H      ; .AND. with mask 1100.000 to find
2560                          ; start of current line...
2570        MOV     L,A       ; Put the masked address back into L...
2580        DAD     B         ;...and add 64 to it = start of next line.
2590        RET               ; via SCROLL 6 CURP
2600 *
2601 *    The printer drivers start at 06CB so we must assemble
2602 *    around them, starting below them at 0711.  The alternative
2603 *    is to also re-assemble the printer drivers, which is
2604 *    not justified for this small program
2605        ORG     0711H
2606 *
2610 ************************************************************
2620 *     FORM FEED ROUTINE
2630 ************************************************************
2640 *                   NOTE
2650 *  Clears the entire current screen and homes the cursor.
2660 *  On Reset or start-up this routine is automacticaly
2670 *  called by CDINIT.
2680 *
2690 FFRTN   CALL    VERTAB    ; Get start and end addresses
2700 FFIN    MVI     M,20H     ; Blank the location
2710        INX     H         ; Advance the address...
2720        CMP     H         ; (A=B4) Are we done?
2730        JNZ     FFIN      ; Nope, keep strokin'
2740 *
2750 *         Read carefully
2760 *
2770 VERTAB LHLD    SCRNEND   ; H = B0, L = B4
2780        MOV     A,L       ; Put screen end address hi-byte
2790                          ; into A - reg
2800        MVI     L,0       ; Set HL to screen home, ie, B000
```



```
Title:  VIDEO

2810        RET
2820 *
2830 *  If we entered VERTAB by a call, we will return via the
2840 *  caller, but if we fell into VERTAB we will return via
2850 *  SCROLL & CURP.
2860 *
2870 ************************************************************
2880 *       MOVER ROUTINE FOR SCROLL
2890 ************************************************************
2900 *                   NOTE
2910 *  This routine is similiar to @TRANSDHBCR.  See page 4-37
2920 * of your Micropolis manual.  The routine will transfer
2930 * -BC bytes,  from the  location  pointed to by  HL
2940 * to the location pointed to by DE
2950 *
2960 REVOM   MOV     A,M       ; Get a character from the location in HL...
2970        STAX    D         ; ...and put it in location in DE.
2980        INX     D         ; Advance destination pointer
2990        INX     H         ; Advance source pointer
3000        INR     C         ; Add 1 to character counter
3010        JNZ     REVOM     ; More?
3020        INR     B         ; Add 1 to character counter
3030        JNZ     REVOM     ; More?
3040        RET               ; No, all done.  Returns to SCROLL
3050                          ; and falls thru to CLEARLINE
3060 ************************************************************
3070 *    SOME BUFFERS...
3080 ************************************************************
3090 *
3100 CURSCHAR DS    1         ; Stores the cursor character
3110 SCRNHOME DS    1         ; Stores screen start hi byte
3120 SCRNEND DS     1         ; Stores screen end lo byte
3130 CURSPOT DS     2         ; Stores current cursor location
3140        END
```

.....

BASIC STATEMENT CROSS-REFERENCE

by Burks Smith

MICROPOLIS BASIC	MICROSOFT BASIC	SYSTEM/34 BASIC	DESCRIPTION
PROGRAMMING STATEMENTS			
--	AUTO	AUTO	Automatic line numbering
DELETE	NEW	CLEAR	Empty program buffer
DELETE	DELETE	DEL	Delete selected lines
DISPLAY	FILES	--	Display disk directory
EDIT	EDIT	n.r.	Edit a line
GOTO, CONT	GOTO, CONT	GO	Restart an interrupted program
LIST	LIST	LIST	List on console
LISTP	LLIST	LISTP	List on printer
n.r.	n.r.	&	Line continuation
LOAD	LOAD	LOAD	Load a program from disk
MERGE	MERGE	MERGE	Merge in program lines
RENUM	RENUM	RENUM	Renumber a program
RUN	RUN	RUN	Run a program
SAVE N:	SAVE	SAVE	Save a new program
SAVE	SAVE	REPLACE	Replace an old program
SCRATCH	KILL	FREE	Delete a file

MICROPOLIS BASIC	MICROSOFT BASIC	SYSTEM/34 BASIC	DESCRIPTION
MATH AND LOGIC			
DATA	DATA	DATA	Internal data definiton
RESTORE	RESTORE	RESTORE	Reset internal data pointer
--	RANDOMIZE	RANDOMIZE	Initialize random nos.
DEF FA	DEF USR	n.a.	Define machine program
DEF FN	DEF FN	DEF FN	Define function
EXEC	--	--	Execute a string
LET	LET	LET	Optional assignment statement
REM, !	REM, '	REM, !	Remark
+	+	+	Add
-	-	-	Subtract
*	*	*	Multiply
/	/	/	Divide
\	\	\	Integer Divide
^	^	¬	Exponentiation
=	=	=	Assignment
--	SWAP	--	Exchange two variables
>	>	>	Greater than
<	<	<	Less than
<>	<>	<>	Not equal
>=	>=	>=	Greater than or equal
<=	<=	<=	Less than or equal
=	=	=	Logical equality
--	--	MAT	Array assignment
OR	OR	OR	Logical OR
--	XOR	--	Logical exclusive OR
AND	AND	AND	Logical AND
NOT	NOT	NOT	Logical inverse
TRANSFER OF CONTROL			
CHAIN	CHAIN	CHAIN	Load and run new program
n.r.	COMMON	USE	Pass variables through CHAIN
n.r.	CALL	n.a.	Call machine program
GOTO	GOTO	GOTO	Unconditional transfer
MICROPOLIS BASIC	**MICROSOFT BASIC**	**SYSTEM/34 BASIC**	**DESCRIPTION**
IF-THEN	IF-THEN	IF-THEN	Conditional execution
ERROR	ON ERROR	ON condition	Error transfer
ON-GOTO	ON-GOTO	ON-GOTO	Conditional transfer
ON-GOSUB	ON-GOSUB	ON-GOSUB	Conditional call
RETURN	RETURN	RETURN	Return from subroutine
FOR-NEXT	FOR-NEXT	FOR-NEXT	Repeat - If loop
--	WHILE-WEND	--	While - Do loop
STOP	STOP	STOP	Terminate program
END	END	END	Optional end of program
TEXT I/O			
PRINT	PRINT, WRITE	PRINT	Direct console output
--	LPRINT	--	Direct list output
INPUT	INPUT	INPUT	Direct keyboard input
--	--	INPUT FIELDS	Read CRT display
n.r.	LINE INPUT	LINPUT	Unformatted keyboard input
OPEN "*T"	--	n.r.	Logical console open
OPEN "*P"	--	OPEN PRINTER	Logical list open
PUT #	--	PRINT #	Logical display output
use FMT	PRINT USING	PRINT USING	Formatted output
ASSIGN	--	--	Logical device assignment
--	--	PRINT FIELDS	Directed output
n.r.	n.r.	FORM	Output format
n.r.	n.r.	IMAGE	Output format
n.r	n.r	PIC	Output format
--	OPEN "O","I"	OPEN STREAM	Open a stream file
OPEN	OPEN "R"	OPEN SEQUENTIAL	Open a sequential record file
OPEN	OPEN "R"	OPEN RELATIVE	Open a random-access file
--	INPUT #	GET #	Read a stream file
--	PRINT #	PUT #	Write a stream file
GETSEEK #	RESTORE	RESTORE	Set sequential pointer
PUTSEEK #	RESTORE	RESTORE	Set sequential pointer
--	--	RETRY	Retry stmt where error occured
GET #	GET #	READ #	Read sequential
--	--	REREAD #	Reread last record
PUT #	PUT #	WRITE #	Write sequential
--	--	REWRITE #	Rewrite last record
GET #	GET #	READ #	Read random
PUT #	PUT #	WRITE #	Write random
n.r.	LSET, RSET	n.r.	Move data to disk buffer
ATTRS	--	--	Set file attributes
STRING	--	--	Set string delimiter
FREESPACE	--	n.r.	Free disk space
n.r.	FIELD	FORM	Define record format
RENAME	NAME	--	Rename a file
EOF	--	--	Set end of file
--	WAIT	n.a.	Wait for port input
CLOSE	CLOSE	CLOSE	Close a file
MICROPOLIS BASIC	**MICROSOFT BASIC**	**SYSTEM/34 BASIC**	**DESCRIPTION**
DEBUGGING FACILITIES			
n.r.	n.r	DEBUG-ON	Turn on debug facility
STOP	STOP	BREAK	Interrupt debugging session
FLOW	TRON	TRACE	Start program tracing
--	--	STEP	Execute one stmt at a time
NOFLOW	TROFF	DEBUG-OFF	Turn off debug facility
OTHER FEATURES			
+	+	&	String concatenation
SIZES	--	OPTION	Set precision
--	OPTION BASE	OPTION BASE	Set array base
--	--	HELP	Help facility
n.a.	n.a.	LIBRARY	Change libraries
--	SAVE ,P	LOCK	Protect program
LINK	--	--	Transfer to object program
IN	IN	n.a.	Input from a port
OUT	OUT	n.a.	Output to a port
--	--	PAUSE	Temporarily suspend execution
PEEK	PEEK	n.a.	Read memory
POKE	POKE	n.a	Change memory
n.a.	n.a.	PROC	Execute a procedure
MEMEND	CLEAR	n.a.	Set end of memory, clear memory
--	SYSTEM	OFF	Return to system

BASIC FUNCTION CROSS-REFERENCE

MICROPOLIS BASIC	MICROSOFT BASIC	SYSTEM/34 BASIC	DESCRIPTION
NUMERIC FUNCTIONS			
ABS(X)	ABS(X)	ABS(X)	Absolute value of X
ATN(X)	ATN(X)	ATN(X)	Arctangent in radians of X
--	--	CEIL(X)	First integer >=X
n.r.	CDBL(X)	n.r.	Convert X to double precision
n.r.	CINT(X)	n.r.	Round X to an integer
n.r.	CSNG(X)	n.r.	Convert X to single precision
COS(X)	COS(X)	COS(X)	Cosine of X radians
n.a.	n.a.	DATE	Date as YYDDD
--	--	DEG(X)	Convert X radians to degrees
EXP(X)	EXP(X)	EXP(X)	Value of e^X
FIX(X)	FIX(X)	IP(X)	Integer part of X
FRAC(X)	--	FP(X)	Fracional part of X
--	--	INF	Largest rational number
INT(X)	INT(X)	INT(X)	Largest integer in X
LN(X)	LOG(X)	LOG(X)	Natural log of X (base e)
LOG(X)	--	--	Common log of X (base 10)

MICROPOLIS BASIC	MICROSOFT BASIC	SYSTEM/34 BASIC	DESCRIPTION
MAX(X_1,X_2)	--	MAX(X_1,..,X_n)	Maximum argument
MIN(X_1,X_2)	--	MIN(X_1,..,X_n)	Minimum argument
MOD(X,Y)	X MOD Y	MOD(X,Y)	X modulo Y
--	--	PI	Value of pi
--	--	REM(X,Y)	Remainder of X/Y
RND(X)	RND(X)	RND(X)	Random number with X as seed
--	--	ROUND(X,Y)	Round X to Y decimals
SGN(X)	SGN(X)	SGN(X)	Sign of X attached to 1
SIN(X)	SIN(X)	SIN(X)	Sine of X radians
SQR(X)	SQR(X)	SQR(X)	Positive square root of X
TAN(X)	TAN(X)	TAN(X)	Tangent of X radians
STRING FUNCTIONS			
ASC(A$)	ASC(A$)	ORD(A$)	Code of A$
CHAR$(X)	CHR$(X)	CHR$(X)	Character whose code is X
n.a.	n.a.	DATE$	Date as "YY/MM/DD"
FMT(X,A$)	--	CNVRT$(A$,X)	String of X formatted by A$
--	HEX$(X)	HEX$(A$)	Hexadecimal representation
INDEX(X$,Y$)	INSTR(X$,Y$)	POS(X$,Y$)	Position of Y$ in X$
LEFT$(A$,X)	LEFT$(A$,X)	A$(1:X)	Leftmost X characters of A$
LEN(A$)	LEN(A$)	LEN(A$)	Length of A$
--	--	LPAD$(A$,Y)	Pad A$ with blanks on left
--	--	LTRM$(A$)	Strip leading blanks
--	--	LWRC$(A$)	Convert to lower case
MID$(A$,X,Y)	MID$(A$,X,Y)	A$(X:X+Y)	Y Chars from A$ starting at X
MAX(X$,Y$)	--	--	Greater string
MIN(X$,Y$)	--	--	Lesser string
--	OCT$(X)	--	Octal representation of X
--	--	PIC$[(X$)]	Return or set currency symbol
REPEAT$(X$,X)	STRING$(X$,X)[1]	RPT$(X$,X)	Repeat X$ X times
--	--	RPAD(A$,Y)	Pad A$ with blanks on right
REPEAT$(" ",X)	SPACE$(X)	RPT$(" ",X)	X spaces
RIGHT$(A$,X)	RIGHT$(A$,X)	A$(LEN(A$)-X:X)	X rightmost characters of A$
--	--	SREP$(W$,X,Y$,Z$)	Search and replace
DISK FUNCTIONS			
ATTR(N)	--	--	Attributes of file N
--	CVI,CVS,CVD	n.r.	Unpack numbers from disk
n.r	EOF	n.r	True if EOF on last access
--	ERL	LINE	Line number of last error
ERR	ERR	ERR	Last error detected
ERR$	--	--	Text of last error message
--	--	FILE(X)	File status
--	--	FILENUM	File number of last error
FREETR(N)	--	n.a.	Free tracks available
n.a.	n.a.	KLN(N)	Key length of file N
n.a.	n.a.	KPS(N)	Key position of file N
n.r.	LOF(N)	--	Length of current "extent"
--	MKI$,MKS$,MKD$	n.r.	Pack numbers to disk
NAME(N)	--	FILE$(N)	Name of file N
--	--	REC(N)	Last record processed
RECGET(N)	LOC(N)	--	Value of sequential get pointer
RECPUT(N)	LOC(N)	--	Value of sequential put pointer
n.a.	--	RLN(N)	Record length
SIZE(N)	--	--	Size of file in records
TRACKS(N)	--	n.a.	Tracks used by file N
n.a.	n.a.	ATTRIBUTE$(X$)	Attributes of display station
n.a.	n.a.	CMDKEY	# of Command Key pressed
--	--	CNT	Count of input items
--	--	CODE	Stop or End code
IN(X)	INP(X)	n.a.	Input from a port
--	LPOS	n.a.	Position of line printer
n.a.	n.a.	MSG$(X,Y$)	Message from a member
PEEK(X)	PEEK(X)	n.a.	Byte from memory
PGMSIZE	--	--	Size of program
--	-POS	--	Position of cursor
n.a.	n.a.	PROCIN	True if input from procedure
n.a.	n.a.	PROCVL	Number of procedures active
SPACELEFT	FRE(X)	n.a.	Remaining memory
--	--	SRCH(array,x,y)	Search an array
--	--	SUM(array)	Sum an array
--	--	UDIM	Un-dimension array
n.a.	n.a.	UPSI$(X$)	Programmable switch setting
--	VARPTR(V)	n.a.	Address of variable
n.a.	n.a.	WSID$	Workstation ID

KEY TO SYMBOLS:

-- The function is not present in the language.

n.a. Not Applicable. Because of operating environment, the function has no value, or can not be implemented.

n.r. Not Required. Because of the internal workings of the language, the function is not required or is performed automatically.

DAMAN PRICE CHANGES

Listed below are the latest prices for software available from DAMAN, with further discounts to the MUG included. I've gone to a two-level pricing. The "normal" price, and a "cash" price. "Cash" means check or money order.

DAMAN, 604 Springwood Circle
Huntsville AL 35803
Phone (205) 881-1697
MDOS S/W PRICE LIST

08/01/82 PROGRAM	LIST PRICE	MUG PRICE	MUG CASH
ACROPOLIS			
FORTH	150.00	153.00	146.12
UTIL	45.00	48.59	46.50
H/W MOD	20.00	21.50	20.00
BONJOEL			
DATABASE TWO	50.00	46.50	44.41
MOD/MATH	50.00	46.50	44.41
INVENTORY ONE	50.00	46.50	44.41
WAMSORT	40.00	38.20	36.48
REACT	50.00	46.50	44.41
PONY-PICK	300.00	254.00	242.57
PONY-PICK II	125.00	108.75	103.86
CHAMELEON			
BALROG	30.00	33.90	32.37
SISYPHUS	30.00	33.90	32.37
MORT FORK	30.00	33.90	32.37

COMPUMAX			
MICRO-LEDGER	140.00	134.30	128.26
MICRO-A/R	140.00	134.30	128.26
MICRO-A/P	140.00	134.30	128.26
MICRO-INVENTORY	140.00	134.30	128.26
MICRO-PERSONNEL	140.00	134.30	128.26
ORDER ENTRY	350.00	322.25	307.75
MAXI-LEDGER	350.00	322.25	307.75
CUSTOM ELECT.			
CCA DATABASE	150.00	133.50	127.49
DISASSEMBLER	50.00	50.50	48.23
PROPERTY MGMT.	70.00	67.10	64.08
DAMAN			
CATALOG	30.00	26.00	24.83
MAIL SYSTEM	50.00	40.00	38.20
MEMBER SYSTEM	50.00	40.00	38.20
DATASMITH			
BOOKKEEPING	250.00	180.00	171.90
PAYROLL	350.00	270.00	257.85
MULMERGE	30.00	26.00	24.83
SMASH	30.00	26.00	24.83
SYSLIST	30.00	26.00	24.83
TEXT CONVERT	75.00	57.50	54.91
VARIABLE LISTER	30.00	26.00	24.83
UTILITY SET	150.00	110.00	105.05
DATA MANAGER	450.00	427.50	408.26
GMS SOFTWARE			
FILE UTILITY	49.00	42.49	40.57
BASIC UTILITY	49.00	42.49	40.57
GAMEDISK	35.00	31.78	30.35
LENZ SOFTWARE			
DISK BANKING	75.00	72.25	69.00
MONK SOFTWARE			
BUILDERS EST.	295.00	254.85	243.38
MICROPOLIS			
MDOS V4	75.00	72.25	69.00
MAINT. MANUAL	50.00	51.50	49.18
ALIGNMENT DISK	50.00	49.50	47.27
DIAGNOSTIC DISK	50.00	49.50	47.27
ORGANIC SOFTWARE			
TEXTWRITER II	125.00	112.75	107.68

SYSTEM/Z			
AUTO/EXEC	40.00	38.20	36.48
BASIC/Z	345.00	291.35	278.24
BASIC/Z MANUAL	35.00	34.05	32.52
BCOMPARE	35.00	34.05	32.52
BEM	65.00	58.95	56.30
CRUNCH	35.00	34.05	32.52
DSM-1	65.00	58.95	56.30
EDIT/S	45.00	42.35	40.44
RUN/Z	65.00	58.95	56.30
SORT/A	75.00	67.25	64.22
TRANSLATOR II	55.00	50.65	48.37
UTL-1	95.00	83.85	80.08
XREF	85.00	75.55	72.15
WORDCRAFT			
MICRO-LINK	89.00	78.87	75.32

Availability and price subject to change without notice. Prices include shipping to N. America. Add $7 per item for shipment (airmailed) outside N.A. VISA & M.C. accepted.

.....

CLASSIFIED

WANTED: CDS Versatile 4. Need not be working, but must be clean and complete. Send history and price.

Dave Montgomery, Box 166, Mt. Pocono PA 18344

.....

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