MICROPOLIS USERS GROUP

MUG Newsletter #24 - July 1982

****

NOTES FROM THE EDITOR

RENEWAL TIME

Take a look at your mailing label. If it says 8207, then this is the last issue of your present membership period. Dues are $18 in North America, $25 elsewhere. Use your VISA or MasterCard, if you wish.

As a results of last month's renewal articles, I've received many comments, positive and negative, on MUG and DAMAN. I'd still like comments from those of you who have yet to renew. Next month, I'll write a summary article on the gist of the responses, and how it will effect our future.

.....

A COMPARISON OF CP/M AND MDOS

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

When I bought my first computer in 1978, it was a Micropolis based system running MDOS and Micropolis Basic version 3.0. I really didn't know much about operating systems, but I wasn't a total stranger to computers. Computers were my minor subject in college ten years earlier so I knew something about programming big mainframes in FORTRAN and assembly language. I hadn't used the knowledge, since the business I was in was too small to afford any computing equipment, until micros came on the market. I bought the MDOS-based system because I was impressed with its capability - and, it was the only computer sold by the only computer store in town.

I was especially impressed with the fact that the operating system and BASIC were interactive. This may seem strange to some, but I remembered that all you did with some of the old computers was give them a stack of cards, press the "LOAD" button, and received either another stack of punched cards, or a lot of printout. That was not interactive at all. I had never heard of CP/M at the time but have since come to know it quite well.

The purpose of an operating system is primarily to perform routine housekeeping functions in the computer and to provide means of running programs. Both CP/M and MDOS perform that mission quite well, but they differ significantly in features and what might be termed "personality." I hope to point out the similarities and differences in a way that are completely fair, but I do have opinions, of course.

FILE TYPES

Both MDOS and CP/M have a system of named files on disk and a way to indicate file type. With MDOS, the type of file is indicated by one of 256 different bit combinations of the type byte associated with each file. Files can be indicated as being one of several different types of object code, source code, BASIC programs, BASIC data, or user-defined, and any file can be either normal, write-protected, permanent, or write protected and permanent.

CP/M codes the file type in the file name. A CP/M file name consists of up to 11 characters, consisting of an 8 character file name, followed by a period, and up to 3 characters as a file type. Some types are recognized by the system as either indicating two types of object code, source code or text, and BASIC programs. All of the rest of the

(Continued in column 8)

BUILDING THE CHEAP COMPUTER, PART III

by Zot Trebor

Gee! I can't believe the response to the first two parts of this thing. Thank you! Both of you! And if the person who wrote in crayon will write again, I'll be happy to answer -- your address was smudged.

When last we met, our hero, the Cheap Computer, was about to put on his Captain Video suit and print something on the screen. For those who didn't see the first two articles, I have used SSM cpu and video cards to make a small computer. The SSM monitor, a whopping 2K of PROM, has proved to be less than useful for even a Cheap Computer. Since the PROM is needed primarily for its video driver routine, my solution to the poor monitor program is to write a video driver just for my cheap computer.

Couple of little problems, here. First off, I've never written a video driver. Secondly, where do I put it once it's written? I can't burn a PROM, which would be the best solution. Can I put it into the RESident module? There's lots of space in RES, but how does one go about it? I don't want to re-write the whole RES module. Lots of little problems.

Let's look at the video driver program first. Just what does a video driver do?

Right off the bat you can see that the SSM video board is nothing more than a crazy kind of memory board. Data stuffed into its 1K of memory will, by dint of electronic genies in their chips (genii?), appear on the video screen. The screen shows 1,024 bytes of data, starting at 0 in the upper left corner, and ending with 1,023 in the lower right corner. Our job is to write a program which will keep track of the current on-screen location, stuff or erase data from it, and scroll everything up one line when the screen gets filled. That takes care of the data, but what about control codes? We need a way to erase the screen when we want to start fresh, which means we need to recognize various control codes and then write little routines to do the special tasks, such as screen clearing.

I'm not a wizard programmer but I've been doing it since 1956. Programming is like chopping wood; experience helps. And like all good programmers, when faced with a new problem, you steal ideas wherever you can find them.

Everyone who makes a video board has also written a driver for it. I managed to locate five or a dozen such programs and studied them. Some, like SSM's, were huge. Others were elegant. All got the job done after a fashion. Scrolling and recognition of control codes seems to be the big time waster. My Cheap Computer uses an 8080 chip at 2 MHz; we don't have time to fool around with the screen all day just to display something. Once the screen is filled you are in effect scrolling for each line so it must be a fairly fast routine.

Why is scroll so important? Let's look at it. We have just filled up the screen with data. Now we want the screen, between one keystroke and the next, to scroll up one line and give us more room. How does it do that? I looked through my Intel 8080 instruction set and couldn't find a single 'scroll' instruction. I read the various programs and discovered that we must actually re-write the entire screen! Seems like a lot of work. We start with line two and write it into line one addresses, then write line three into line two and so on. The last line; the one on the bottom of the screen, we simply erase or fill with blanks. Line one flys off the screen into the bit bucket and by the time our finger hits the next key, line sixteen is clear and ready to receive our input. Hey, this isn't so tough after all.

(Continued in column 11)

DATASMITH MDOS UTILITY PACKAGE

This review is the second of a series of articles on the available MDOS utilities. (See June newsletter for Acropolis and GMS utilities.)

The DATASMITH utility package for MDOS includes a number of programs useful to programmers writing applications in Micropolis Basic. In addition, several general-purpose utilities have been provided that are of general interest to users of the MDOS operationg system. For maximum utility to the user, 8080 source code has been provided to permit user modification of the programs for special uses, if desired. Again, as with the Acropolis utilities,study of this code will give you valuable insight on the MDOS system and the use of the built-in subroutines.

The programs are also provided in executable object files, so you can run them without worrying about editing and assembling source files.

Specifically, the package consists of 9 programs, each being discussed below. The total package is available from DAMAN for $116, or separately as:

BASIC VARIABLE LISTER $27
BASIC TO LINEEDIT & LINEEDIT TO BASIC $60
MULTIPLE MERGE $27
BASIC SYSTEM LISTER $27
BASIC PROGRAM SMASH $27

DUMP MEMORY, DUMP FILE, and the PHYSICAL DISK DUMP are not available outside the package. The prices are postpaid to North America. Add $5 airmailed elsewhere. VISA & Master Card accepted.

INSTRUCTIONS FOR USE

The following paragraphs describe the operation and syntax of each program on the development package disk. Following the established Micropolis conventions,information in <> symbols indicates parameters that are to be provided by the operator. Information in [] symbols indicates optional parameterswhich, if entered, override "default" values assumed by the program. In both cases, the <> or [] symbols are not part of the command and should not be entered. Information not inside these symbols must be entered. A default drive number of 0 is assumed if no unit number is entered.

BASIC VARIABLE LISTER

The BASIC VARIABLE LISTER lists, on the console, all the variables and dimensioned arrays used in the Micropolis Basic program referenced by <filename>. If the optional parameter [1] is included, the program leaves a blank line in the listing for letters that have not been assigned as variables. Command syntax is:

VARLIST "[unit:]<filename>" [1]

Limitations: The program includes user-defined function names from DEF FNx or DEF FAx commands as variable names. It will report dimensioned arrays with calculated dimensions, but the dimension itself may appear as garbage. If printer output is desired, console data must be sent to the list device with the ASSIGN command.

BASIC TO LINEEDIT CONVERSION

The BASIC TO LINEEDIT CONVERSION converts the Micropolis Basic program file referenced by <Basic file> to a LINEEDIT compatible text file of the name <text file>. The program expands one-byte Basic command and function tokens to ASCII text and generates LINEEDIT line numbers with proper compatibility. This program is useful when a text copy of a Basic program is needed for transimission via telecommunications or when global search and change features of LINEEDIT are desired.

(Continued in column 6)

ONE VIEW ON FORTH

by George W. Shaw II
Shaw Laboratories, Limited

What is Forth? It has been said that Forth is the first in a new generation of programming languages, but what does this mean? How are they different? What makes them a new generation of languages? This is what I will try to explain.

Conventionally, computers are programmed in an artificial language, that is, a language which is neither natural to the computer nor natural to the programmer. The language is not natural to the computer in that it does not directly relate to the computer's own machine instruction set, and it is not natural to the programmer in that it is not English or whatever his native language may be. Because of this, a programmer can never perform his task directly; he usually has to work within the confines of the given artificial language to solve his problem. This can cause great inefficiancy in both the programming of the problem and the execution of the problem by the computer.

Forth attempts to transcend this extra level. Rather than creating an artificial language, Forth first attempts to perfect the computer's language. The very basic Forth system is then actually a set of uniform computer instructions. This basic set of instructions is referred to as the kernel, or nucleus, of the system. Many of the instructions in the nucleus duplicate actual processor instructions. Other instructions which are not available are defined as short code routines for implementation. This uniform set of instructions, in effect, creates a pseudo computer; an instruction set which actually does not execute directly on any existing machine. This group of instructions, the nucleus, is called the Forth Virtual Machine.

The Forth Virtual Machine gives the programmer the ability to program the computer as efficiently as possible by programming almost directly in the machine's native language. The efficiency of the Virtual Machine will vary from computer to computer, depending upon its actual native instruction set. On a typical 16 bit implementation, the overhead is usually only about 20% above direct native code; on an 8 bit it is about 100%. The advantage of the Forth Virtual Machine over programming any processor directly is that the entire programming process is under the control fo the Forth System. Programming, Testing, and Debugging may be done interactively from the keyboard. This allows much greater efficiency than programming in the usual type of assembler system. Also, because the Forth Virtual Machine is very uniform, there are fewer idiosyncrasies for the programmer to be concerned with.

Upon this Virtual Machine are then built the tools which are necessary for the application. Many tools are already available in the system because they are necessary for the compilation and interpretation of Forth text. The system may be built upon and extended as additional commands are needed to complete an application. This allows the programming language to be tailored directly for the language of the application, rather than the application's language being translated into an artificial programming language. It is simply impossible to describe all the problems of the world in a word set and structure as limiting as those of BASIC, Pascal, FORTRAN of any other conventional programming language.

Once the basic language for the application has been implemented, the upper levels of the application are programmed directly in this language. Forth also allows the programmer to program directly in the machine's native code just as he would program in high level Forth itself. This allows the mixing of machine code and high level Forth in any manner desired. Because of

(Continued in column 13)

WRITING PACKED RECORDS - Part III

by Buzz Rudow

In the March and April issues of the MUG newsletter, I discussed the topics of packing the Micropolis 250-byte physical sector with two 125-byte logical records. Inherent in this application is the absolute positioning of data, so that delimiters are not used. If interested, you had best review the previous article before going any further here, as I'll try not to repeat myself. I promised to bring it all together in May, but I didn't. Now I will.

Following the text of this article is the code for the "Input and Modify" routine, one of 21 modules used in the mail system sold by DAMAN. It uses the auto-configuration and INKEY routines shown in previous newsletters. These routines are set up in the Main Menu.

Using structured programming, the entire module is referenced in lines 30-50. Since the purpose of this article is to discuss the disk accessing, I'll not explain anything but that topic. The rest is there for you to see, and I'd be glad to individually discuss any details with those of you interested.

To get to he point of interest of the program, we trace from line 50 to subroutine 40060. If the operator answers "1" to that menu, line 40210 takes you to subroutine 10015.

PROBLEM OF PARTIAL PHYSICAL RECORDS

One problem with packing two logical records to a physical record is that the operator may stop inputting data in mid-physical record. If that happens you must:

(1) pad bytes 126-250 with blanks
(2) make sure you write it out to the disk

Since you can now have this "half-empty" record, you also must test for the condition whenever operator input is resumed on an existing file.

A SOLUTION

First, let's look at the case of a new file. If the operator answers "new file", or "1", then lines 10035 - 10055 are executed to open the file, and then jump to line 10080. Lines 10110-10125 take the input generated and write it to disk. It's simple enough. The variable that keeps track of what side of the physical record you are entering is "I%". It is set to "1" in line 10030. If I%=1, which is the case of a new file, then the program saves the input in the left side of Z$, increments I%=2, and returns. If I%=2, which may be the case for an old file, and which will be the case if you just saved a first entry, as above, then the program appends the input to the right side of Z$, writes it ("it" being both entries) to disk, and sets I% back to 1.

Line 10090 is the test for end-of-input by the operator. If the operator quit, then the length of A$=0. If so, the program skips to line 10130. Here we check the value of I%. As we just discussed, if I%=1, then there is no input being held in Z$. If I%=2, then Z$ holds a 125-byte logical record that hasn't been written to disk. Line 10130 tests I%, and if equal to 2, puts 125 bytes of blanks on the end of Z$ and writes all 250 bytes to disk.

Now back to the problem of re-starting input with a partial physical record. If the operator answers "old file", the program asks for, and opens, an existing file, then jumps to line 10060. Here I read the last record and test for a blank in character position 126. There will never be a blank in 126 if the right-hand logical record exists, as the system always requires a minimum of a NAME to be entered.

(Continued in column 14)

DATASMITH UTILITIES - (Continued from column 3)

Command syntax is:

BAS>LIN "[unit:]<Basic file>" "[unit:]<text file>"

Limitations: Basic Programs allow a line length of up to 250 characters, while LINEEDIT only allow a maximum of 132 characters. Therefore, it is impossible to convert a basic line exceeding 132 characters to LINEEDIT format. If any line in the Basic program exceeds 132 characters, the program will abort with a PARM ERR message. It is recommended that long lines in the Basic program be broken in smaller lines or deleted before performing the conversion. Note that LINEEDIT treats everything as text and line references in a Basic program have no meaning to it.

LINEEDIT TO BASIC CONVERSION

The LINEEDIT TO BASIC CONVERSION converts a LINEEDIT text file referenced by <text file> to a Micropolis Basic program file referenced by <Basic file>. The program compresses Basic command and functions in ASCII to one-byte tokens and strips LINEEDIT line numbers. The program is useful to recover files that had been previously converted to text by the BAS>LIN program and has many useful applications in converting programs from other languages to Micropolis Basic. Command syntax is:

LIN>BAS "[unit:]<text file>" "[unit:]<Basic file>"

Limitations: This is a "dumb" conversion program and it assumes that the original text file is of proper Basic syntax and all line numbers are correct and in ascending order. It will try to convert any file produced by LINEEDIT, if asked, but the results may contain garbage that Basic can not recognize. Make sure that your source file is correct before using this program.

MULTIPLE MERGE PROGRAM

The MULTIPLE MERGE PROGRAM merges a common basic program segment in up to ten different Micropolis Basic files without operator intervention. When the program signs on it prompts:

ENTER MERGE FILE NAME?

whereby the operator is expected to enter the name of the Merge file in the form: "[unit:]<filename>". Note that the file name must be enclosed in quotes. After the Merge file name has been entered the program will prompt:

ENTER A PROGRAM NAME?

whereby a program name in quotes is expected. Pressing RETURN without entering a file name will begin program exectuion. This program serves the same function as several LOAD, MERGE, and SAVE commands from Basic and is useful for configuring systems consisting of several programs with hardware-dependent code, serial numbers, etc. Command syntax is:

MERGE (no parameters)

Limitations: This utility saves the entire program in memory during the merge, so there must be ample memory for both the program and the MERGEd code. The program does not check for a memory overflow and will crash the system if it is asked to MERGE a program that is too long. The utility is quite short, however, and generally has the ability to MERGE programs that are longer than Basic could load on the same system, so this is seldom a problem.

BASIC SYSTEM LISTER

The BASIC SYSTEM LISTER prompts for up to ten names of Micropolis Basic program files. The files must be entered in the form "[unit:]<filename>", including quotes. Pressing RETURN without entering a file name starts execution of the program. Each program is listed on the system printer with the

name of the program and a page number as a heading. New programs start at the top of a new page. Equivalent to several LOAD and LISTP commands from Basic, this utility allow listing several programs without operator intervention or printer "babysitting". Command syntax is:

SYSLIST (no parameters)

Limitations: The program assumes a standard 66 line computer form of 132 characters width. The program would have to be modified to handle other sized forms.

BASIC PROGRAM SMASH

The BASIC PROGRAM SMASH ROUTINE removes remarks and non-significant spaces from the program referenced by <source file> and creates a "smashed" version as the file <destination file>. This utility is useful for reducing the memory requirements of a Micropolis Basic program that contains REM or ! statements and spaces that make the program easier to read. The resulting program is difficult to read, but is of the smallest size that will still execute. No line numbers are removed, so remarks may still be used as entry points in the program. Command syntax is:

SMASH "[unit:]<source file>" "[unit:]<destination file>"

Limitations: Since the resulting code is difficult to read, use this only for debugged programs, and save the fully commented form for reference. In some rare cases, removing spaces may produce ambiguous statements that may cause syntax errors in the resulting program. For example, the statment: PUT 2 R, S, T when "smashed" becomes PUT2R,S,T. Basic would interpret the 2R in the "smashed" version as a lead-in to a file number expressed in Radix 2 (binary) format, but when the first comma is encountered it assumes it has found a syntax error. The meaning can be made clear simply by inserting a single space after the file number. This is the only type of error that has been encountered as a result of using the program.

DUMP MEMORY

The DUMP MEMORY program is an enhanced version of the MDOS DUMP command which dumps memory to the console in ASCII representation as well as hexadecimal. All bytes that could be interpreted as valid ASCII characters are printed, and bytes which do not fall in the range of ASCII characters are printed as periods. Command syntax is:

DMEM <start addr.> <end addr.>

Limitations: Since this is not a resident MDOS command, calling it will overlay a small part of the applications area.

DUMP FILE

The DUMP FILE program is an enhanced version of the MDOS DISP command which dumps the contents of a disk file to the console in ASCII representation as well as hexadecimal. The output format is the same as the DMEM program. Command syntax is:

DFILE "[unit:] <filename>"

Limitations: None discovered.

PHYSICAL DISK DUMP

The PHYSICAL DISK DUMP UTILITY is similar to the DFILE program, except it dumps the content of a Micropolis disk by physical track and logical sector instead of by file name. This allows viewing "system" areas of the diskette such as the trackmap and directory, and will also allow viewing the contents of different formats such as CP/M for Micropolis diskettes. Command syntax is:

DISKDUMP (no parameters)

This is a command-oriented program with four commands as follows:

DRIVE <unit>
Specifies the drive number to access. The default is 0. Once the drive command has been used, the drive number remains the same until another DRIVE command is given.

TRACK <track #>
Specifies the physical tack number in hexadecimal notation that will be accessed on the next read. Once a TRACK command has been given, the track number remains the same until another track command is given.

SECT <start sector #> [end sector #]
Causes a physical disk read with the desired range of sectors (entered in hexadecimal) to be displayed on the console in the same format as the DFILE program. The second sector number is optional. If it is not entered, only one sector will by displayed. These are "logical blocks" rather than physical sectors. See the Micropolis MDOS manual for details on sector mapping.

HELP
Displays the command syntax on the screen. The help display is called on sign-on or syntax errors.

DOS
Terminates the program and returns control to MDOS.

Limitations: You have to know what you are doing for this program to be much use.

.....

CP/M & MDOS - (Continued from column 1)

CP/M file types are user-defined and can be any combination of 3 characters or less.

User file types are routinely used with CP/M because they are part of the file name, and handy features for treating files as groups are provided. Through the use of the CP/M "STAT" program, a separate file on disk, any file can be tagged as read/write or read only, but no equivalent to the MDOS "protected" type exists. CP/M also has a file type (SYS) that doesn't appear in directory searches and only the "STAT" program can find them. This is an interesting but seldom used feature.

CP/M has a way of logging on "users", so that each user has his own directory and can share a disk with another "user", never seeing each other's files. The "user" idea is great for winchester disk systems but isn't used much with floppys because their limited storage makes more than one user impractical. MDOS actually pays more attention to file types as far as the system is concerned, while with CP/M they are primarily for the user to define.

SYSTEM TRACKS

Under CP/M, the operating system itself resides on the disk on two reserved tracks separate from the rest of the data on the disk. A utility program called SYSGEN is required to put it there, and CP/M does not appear on the disk directory. In fact, it is impossible to tell if CP/M is on a disk without actually trying to "boot" it. Under MDOS, everything except the bootstrap loader is just another file in the directory and the filecopy utility can be used to transfer MDOS from disk to disk.

CHANGING DISKS

One of the more annoying features of CP/M is the fact that you can't change the disk in any drive without the new disk being tagged "read only" by the operating system. Any time you do perform a disk swap, you must enter control-C to do a "warmboot" of the system, which amounts to CP/M's console command processor being re-read from disk and all disks being tagged read/write again. This means, of course, that CP/M must be on the disk in

the default drive or the system will crash with a "system error". I would assume this "feature" is there to protect the user from errors caused by indiscriminate disk swapping, but it turns out to be destructive in most cases because it prevents you from changing disks even when it is beneficial to do so.

MEMORY ALLOCATION

Both systems provide very similar ways of running programs by means of simply typing an executable file name followed by a number of optional parameters. For example, to run Basic in either system, you just type BASIC. The program is loaded into an applications area and executed in either case, but memory is managed differently in each system.

With MDOS, the operating system resides in low memory and programs are loaded in at a higher memory address. MDOS can use all contiguous memory available in the computer and automatically determines the location of the end of memory.

CP/M resides in high memory and loads programs in low memory, usually at 100 Hex. A utility program called "MOVCPM" can relocate the operating system anywhere in memory and must be used to install CP/M on computers with different memory sizes. This slight inconvenience is offset by the fact that programs running under CP/M are somewhat more transportable, all using the first 256 bytes of memory as an interface to CP/M no matter where it may be in memory or what version it is. It allows CP/M itself to be of different physical sizes without any disruption of application programs. Since MDOS version 4 is larger than version 3, the address where application programs are to be executed had to be changed to make room for it.

SAVING MEMORY

Both CP/M and MDOS have a SAVE command whereby a memory image can be saved on disk, but the CP/M SAVE is severely limited, only having the ability to save memory starting at the beginning of the applications area at 100H and proceeding for as many 256 byte "pages" as specified. MDOS, on the other hand, can save memory between any two addresses, making it considerably more versatile. CP/M has no ability to load a program or any memory image without executing it, while MDOS can load a file to any memory location and even do "scatter loads" where a single file can be loaded into non-contiguous locations.

MEMORY DUMP

All of the commands present in MDOS for memory operations are absent from CP/M. Memory dump, block moves, searches, fills, compares, and several other programmer-oriented utilities can not be done under CP/M unless the separate debugging program is loaded, or your computer has a monitor program in ROM that performs these functions. I appreciate the MDOS utilities for development purposes, but have never known a commercial computer user to need them or even care if they were there.

APPLICATION INTERFACES

Like MDOS, CP/M has a number of entry points that allow application programs to use the operating system's input-output interfaces. However, the CP/M interface functions are very basic in nature and do not allow anywere near the variety that MDOS does. For example, MDOS supports automatic byte-for-byte reading and updating of a file, block loads and saves, command line parsing, error handling routines, processor utility routines, and several extremely useful conversion routnes, like ASCII to Binary conversion, 16-bit multiply and divide, etc. The fact that these functions are preprogrammed into the operating system makes assembly language programming considerably easier with MDOS.

Also, CP/M does not return any specific error information to an applicaiton program. When doing a disk operation, CP/M only indicates whether or

not the operation was successful to application programs, and only a "bad sector" message to the operator. MDOS has nineteen different error codes.

Consider the following typical scenario: You have just spent an hour with your text editor writing a letter or an assembly language program. You don't know it, but there isn't enough room on your disk when you enter the command to save it. Under MDOS, the operating system will report "DISK FULL" and return you to the editor. You may now put a disk with more room on it in the drive and try again, with no data lost.

Under CP/M, there is no "DISK FULL" message, so the system will report a "DISK ERROR" with no clue as to what might be wrong. Also, you end up with the operating system in control, with no way to get back to the text editor or your hour's worth of work. Even if you aren't thrown out of your text editor, you can't swap disks because CP/M will mark the new disk "read only".

What do you do? If you are Joe Average user you might utter a few choice words or shed a few tears, but there is no "documented" way to recover from this common problem under CP/M. Actually, there is a way you can get your data back by tricking CP/M or using your monitor (if you have one) to move the data down in memory so CP/M can save it. This requires an intimate knowledge of both your computer and CP/M, and an unsophisticated user wouldn't have any idea as to what to do.

While MDOS has the edge in power, CP/M has a formal interface scheme that makes programs much more portable. MDOS function entry points are memory addresses containted in EQU statements in the files "SYSQ1" and "SYSQ2". This means that the programmer has to have access to the user's function entry points in order to deliver a running assembly language program under MDOS. CP/M uses a "function code" and a call to a single low memory address for all functions. This assures that programs designed to run under CP/M will be transportable from one system to another, and that modifications to the operating system will not affect the programs that run under it if addresses have to be changed.

UTILITY PROGRAMS

Both operating systems have a number of utility programs associated with them as "standard equipment". CP/M relies on utility programs more than MDOS, since many functions that MDOS does standing alone are done by CP/M utility programs. For example, CP/M uses a program called "STAT" to perform some of the functions that TYPE, FILES, and ASSIGN do under MDOS. STAT has indespensible features that enhance CP/M greatly, but since it is a separage program, it can not be used while any other application is running. CP/M does not have a FREE command that returns the amount of free space on a disk, and there is no simple way a program can tell it is running out of disk space. CP/M only tells you after you've already run out.

The Assembler provided with CP/M is a good, reliable 8080 assembler that is comparable to the MDOS assembler. Both have about the same features, except that the CP/M assembler produces object code in Intel Hex format rather than in loadable and executable code. For those not familiar with it, Intel Hex format is a system of representing object code as an ASCII representation of hexadecimal numbers for addresses, op-codes, and data. Since it contains only ASCII characters, a Hex file can be printed on a printer or sent from computer to computer via a modem. Another CP/M program called "LOAD" creates an executable file from a Hex file. Output from the MDOS assembler can be either in memory or in a file that can be directly loaded and run.

Both systems have editors to produce text files either for the assembler or for general purpose text processing. The editors are line oriented rather than screen oriented and therefore are rather crude when compared to word processing

software. While the features of both are comparable to a certain extent, the CP/M editor, ED, is much harder to use than the Micropolis LINEEDIT editor, in my opinion.

The debugging programs provide needed support for developing assembly language programs. The CP/M version, DDT, self-relocates to the highest available memory area, while Micropolis DEBUG has a DEBUG-GEN program that produces a debugger that will run anywere the user selects. Both have similar features, but CP/M relies on DDT to perform certain functions that are built into MDOS.

By far the most useful program with CP/M is the Peripheral Interchange Program, or PIP. Using PIP, you can connect any device to any other device in the system and perform a data transfer. PIP is used most often for copying files, like the Micropolis FILECOPY, but PIP is much more versatile. Not only can you connect any device to any device, but PIP will also perform some editing like line and page numbering, lower case to upper case translation, expanding tabs, etc. on the way. CP/M can support more devices than MDOS and PIP can perform a wide variety of transfers. Besides disk-to-disk, PIP can transfer disk to screen, printer, or one of several user defined devices, and device to disk, even keyboard to disk.

CP/M also has a very powerful batch-processing feature called SUBMIT that allows commands contained in a text file to control the computer in lieu of the keyboard. With SUBMIT, the user can program the operating system to perform a whole series of operations like copying files, running programs, making listings, etc. and then walk away while they are all executed in sequence. As soon as one job is over, the next in the queue is performed automatically. This is a great time-saving feature that has no counterpart in MDOS.

SUMMING IT UP

In summary, both systems have arguable strengths and weaknesses. Because MDOS specializes in one type of disk drive and is a much larger system than CP/M, it is much more friendly to the user and allows programs to be written that can handle almost any error under software control without "crashing." CP/M is written so that it can be customized for almost any type of disk storage system and therefore doesn't really know much about the hardware it is controlling. As a result, its simplistic approach can easily result in a program being aborted by a simple thing like a disk not being in the drive. It is CP/M's simplicity, however, that has made it successful because it is easily adapted to any 8080 or Z-80 based computer.

If you are looking for the widest possible variety of software to run on your computer, you have little choice but to run CP/M since more software has been written for CP/M based systems than for any other operating system. However, if you have Micropolis equipment and can find the MDOS software you need or are developing high quality programs for a user that doesn't need access to all those CP/M programs, MDOS will make the programming chore much easier and is many times more friendly to the user. You can always run CP/M on a Micropolis system if you need to, but you can't run MDOS on anything but Micropolis equipment. This gives Micropolis owners the option if having two operating systems if they need them.

I personally would like to see MDOS modified to include some of the desirable features of CP/M, and perhaps even with new disk drivers so that it could be transported to non-Micropolis equipment. I would be interested in hearing from readers that have thoughts on this subject.

.....

CHEAP COMPUTER - (Continued from column 2)

Gosh, I like computers! Getting to re-write everything on the screen just to enter the next line. You'd think it would know or something...

Alright, I think I can do the scroll. What about the control codes? We have to know if it's a printing character or a control character. And what about backspace? ASCII backspace is code 08 while Micropolis insists on 5F. Ummm.

The most common way of deteriming what a character is, uses the Compare Immediate instruction, CPI x. Whatever we make x, it will be compared to the contents of the A-register. If they match, the zero flag is turned on. Kind of clunky, but it works. Actually, all we want to do is turn on the zero flag. We don't care if it gets raised because of a CPI, an SUI (Subtract Immediate), an ADD or an attack on the Falklands. We can start with SUI 0D. If the A-reg contained 0D then the flag turns on. We can then increment the A-register. The flag will turn on when the A-register equals zero, and that will tell us what it originally contained. This is faster than a string of CPI's.

Perhaps this isn't very clear. In the coding I've added a lot of comments so you can follow it. If you are only doing one or two compares, then use CPI, but if you are working with a large number of control codes, the method above is faster.

When we get right down to it, characters coming to the video driver can only do two things; they can add space to the screen, like a backspace or screen clear, or they can subtract space from the screen, like a normal printing character or a line feed. Anything else, we simply ignore. But the control code recognition routines give us a very handy place to recognize codes and branch to non-video related routines, like Bell or Printer control. I've added some comments about this in the coding and for those of you who want to personalize your system; to modify it to meet your specific needs, the video driver is a nice place to handle your special codes. Read the code, it's very flexible.

A SHORT LECTURE

Is it even practical to waste our precious MUGger Newsletter talking about 16 x 64 video displays and cheap computers when most business applications use 24 x 80 displays? I think the jury is still out. The smaller display has marked disadvantages but it is also lower in cost and for second and subsequent systems, devoted entirely to data entry or other mundane chores, it is perfectly adequate. The 'Cheap' philosopy; using seperate cpu, video and I/O boards actually has a lot to say for it. In my area (San Diego) the sophisticated systems have quickly outstripped the ability of the local technicians to maintain them; if it doesn't work it doesn't matter how sexy the system is. In my Cheap Computer the costs are low enough that I can keep seperate back-up cards and simply swap cards if a problem developes. (But the hardware is well burned in and I've had no equipment failures in more than a year.)

END OF LECTURE

Next time I come around on the disk drive, I'll offer you a nice, neat little video driver, suitable for any 16 x 64 memory mapped video board, plus the instructions on how to install it without causing a nervous breakdown. For you hardware buffs, light a fire under your soldering iron... what? An Electric Soldering Iron! What'll they think of next! Well, get it out anyway, because I'm going to show you how you can give your Cheap Computer a bell... or at least a horn. You'll need a parallel outport and for fun, I'll show you how to play music on your bell. God willing.

.....

VIEW ON FORTH - (Continued from column 4)

Forth's efficiency and speed of execution, machine code is usually only resorted to when execution time is an extremely critical factor.

Forth attempts to create the concept of a more perfect programming environment. First, by creating the Virtual Computer in its nucleus instruction set and then, by allowing the programmer to create the commands which are most useful for an application. The programming language is customized for the application rather than the application being tailored for the language. This allows a more direct implementation of the problem into a program. If the problem can be conceived of as a six headed dragon, then the language can be structured to appear to manipulate a six headed dragon. But this is only the beginning. For the programmer there are many more features.

BASIC is a good programming language because, among other things, it is interpretive in nature. That is, segments of a program can be individually tested at the keyboard by the programmer. Forth can do this too, and even more. By their very nature, good Forth programs are very modular. In fact, each module can be accessed individually and tested. Additionally, because Forth passes data on the stack and not through variables, the programmer does not have to worry about accidentally reusing the wrong variable, or what the variable Z7 or X9 was used for. Forth has variables (though they are used much less) which may have names of any length desired (31 characters in most popular implementations). So cryptic names such as Z7 and X9 are no longer a problem.

Forth also does not have the memory problems of other languages. First, the language itself is very small, smaller in fact than most extended BASICs and operating systems. In fact, Forth code is so small that much of Forth is written in Forth. Except for the Virtual Machine and a few other time critical Forth instructions, all of the Forth system including the interpreter, compiler, editor, and assembler are written in Forth. Also, all of mass storage (disk for example) can look to Forth like virtual memory. This allows a program to use strings or arrays as large as all the mass storage available, just as if they were in memory. For additional savings, Forth can even be programmed to chain or overlay programs when necessary. In fact, Forth is the only language efficient enough for Hand-Held Computers. Except for Microsoft BASIC available on the Quasar or Panasonic HHCs, almost all of the programs and languages on the HHCs are written in a version of Forth.

Forth is faster than most languages, even languages which compile. Forth both interprets and compiles. This gives it the programming flexibility of BASIC but the speed of compiled languages. When compiled, Forth takes much less space than other compiled languages. This is because Forth compiles threaded code. That is, rather than compiling the actual native processor instructions to be executed (often several bytes for a single command), Forth compiles a pointer to the code to be executed. These pointers are executed in sequence by a small code routine called NEXT. This routine is very short (about ten bytes) and very fast (about 3/8 the speed of a subroutine call) and is one of the keys to the flexibility of Forth.

Just as a good BASIC programmer can write very good BASIC code and a bad BASIC programmer can write incomprehensible BASIC code, the same is true of Forth programmers, but to a greater degree. Forth is much more of an amplifier of a programmer's talents because the language does not protect the programmer from himself. This allows a good Forth programmer to write very short, concise, correct, program code and a bad Forth programmer to write long, strung-out, code which may never actually work and looks more like FORTRAN than Forth. Many people do not, and will not, like the language for this reason. Forth requires a programming philosophy and style different from most languages. When programmed properly, however, it is the most

efficient language for small systems. Do you then dare to attempt to program in Forth and possibly learn the level of your talents and understanding?

.....

PACKED RECORDS - (Continued from column 5)

If character 126 is a blank, then Z$ is set to the left-hand 125 characters of the current Z$, I% is set to 2 (to say I have a partial physical record), and PUTSEEK is decremented by one. (When you open a file, PUTSEEK=RECPUT, i.e., one bigger than SIZE, being the next location to write a sequential record.) This means I'll write over my current last record - after I fill the vacent right-hand side.

Lines 12060-12085 show a method for determining where a logical record is on the physical file.

You'll note that I've tried to get all operator response to be a string input, though, as I review the listing, I see I haven't succeeded. I believe in checking all responses for validity before trying to execute on them, such as in lines 3025-3030. Line 11020, however, takes an integer response. It shouldn't. The code should look like that at lines 12025-12045.

The listing is by System/z' BEM, by the way. Very helpful. The domain of each subroutine is shown by the "*'s", with the entry point shown at ">*" and the exit at "<*". Locations of "GOTO's" are shown by the "*>" or " >". The print of variable and space allocation is also done by BEM.

.....

Title: MAIN

```
   10      GOTO 30: ! 02/28/82
   20      SAVE "MAIN":PRINT "SAVED `MAIN`":END
   30    > GOSUB 57215: ! Test for V3/4
   35      GOSUB 57115: ! Read Clear Screen
   40      GOSUB 31120: ! Configure
   50      GOSUB 40060: ! Main Menu
   60      PLOADG "MENU.M"
  500      !
  505      ! 05/09/81 Keyboard Input
  510      !
  515 >*   PRINT O$(0)
  520   *  PRINT "Press RETURN for any field not use
           d."
  525   *  PRINT
  530   *  PRINT "ENTER NAME or Press RETURN to Exit
           "
  535   *  PRINT L$
  540   *  INPUT A$
  545   *  IF LEN(A$)=0 THEN 735
  550   *  A$=A$+M$
  551   *  PRINT
  555   *  PRINT "ENTER ADDRESS 1"
  560   *  PRINT L$
  565   *  INPUT B$
  570   *  B$=B$+M$
  571   *  PRINT
  575   *  PRINT "ENTER ADDRESS 2"
  580   *  PRINT L$
  585   *  INPUT C$
  590   *  C$=C$+M$
  591   *  PRINT
  595   *  PRINT "ENTER CITY"
  600   *  PRINT LEFT$(L$,15)
  605   *  INPUT D$
  610   *  D$=D$+M$
  611   *  PRINT
  615   *  PRINT "ENTER STATE"
  620   *  PRINT LEFT$(L$,4)
  625   *  INPUT E$
  630   *  E$=E$+M$
  631   *  PRINT
  635   *  PRINT "ENTER ZIP"
  640   *  PRINT LEFT$(L$,11)
  645   *  INPUT F$
  650   *  F$=F$+M$
  651   *  PRINT
  655   *  PRINT "ENTER COUNTRY"
  660   *  PRINT LEFT$(L$,4)
  665   *  INPUT G$
  670   *  G$=G$+M$
```

```
  671   *  PRINT
  675   *  PRINT "ENTER AREACODE"
  680   *  PRINT LEFT$(L$,5)
  685   *  INPUT H$
  690   *  H$=H$+M$
  691   *  PRINT
  695   *  PRINT "ENTER PHONE"
  700   *  PRINT LEFT$(L$,9)
  705   *  INPUT I$
  710   *  I$=I$+M$
  711   *  PRINT
  715   *  PRINT "ENTER FLAG"
  720   *  PRINT LEFT$(L$,7)
  725   *  INPUT J$
  730   *  J$=J$+M$
  735 <*> RETURN
 1100      !
 1105      ! 05/09/81 Display to Screen
 1110      !
 1115 >*  PRINT O$(0)
 1120   *  PRINT "A - NAME: ";A$
 1125   *  PRINT "B - ADD1: ";B$
 1130   *  PRINT "C - ADD2: ";C$
 1135   *  PRINT "D - CITY: ";D$
 1140   *  PRINT "E - STAT: ";E$
 1145   *  PRINT "F -  ZIP: ";F$
 1150   *  PRINT "G - CNTY: ";G$
 1155   *  PRINT "H - AREA: ";H$
 1160   *  PRINT "I - PHON: ";I$
 1165   *  PRINT "J - FLAG: ";J$
 1170   *  PRINT
 1172 <*  RETURN
 3000      !
 3005      ! 05/09/81 Modify
 3010      !
 3015 >*> R$=""
 3016   *  PRINT "ENTER 'LETTER''SPACE''CORRECT TEXT
           '"
 3017   *  PRINT "OR, Press RETURN to Continue."
 3019   *  INPUT R$
 3021   *  IF LEN(R$)=0 THEN GOTO 3095
 3025   *  IF LEFT$(R$,1)<"A" OR LEFT$(R$,1)>"J" PRI
           NT "ERROR*****:FIRST CHARACTER MUST BE 'A
           ' THRU 'J'":GOTO 3015
 3030   *  IF MID$(R$,2,1)<>" " PRINT "ERROR*****:SE
           COND CHARACTER MUST BE A BLANK":GOTO 3015
 3035   *  S$=LEFT$(R$,1)
 3040   *  T$=RIGHT$(R$,LEN(R$)-2)+REPEAT$(" ",28)
 3042   *  N%=N%+1: ! Count Mods
 3045   *  IF S$="A" A$=T$
 3050   *  IF S$="B" B$=T$
 3055   *  IF S$="C" C$=T$
 3060   *  IF S$="D" D$=T$
 3065   *  IF S$="E" E$=T$
 3070   *  IF S$="F" F$=T$
 3075   *  IF S$="G" G$=T$
 3080   *  IF S$="H" H$=T$
 3085   *  IF S$="I" I$=T$
 3090   *  IF S$="J" J$=T$
 3095 <*> RETURN
 3600      !
 3605      ! 05/09/81 Separate Fields of incoming Re
           cord
 3610      !
 3615      IF I%=1 Y$=LEFT$(Z$,125)
 3620      IF I%=2 Y$=RIGHT$(Z$,125)
 3625 >*  A$=LEFT$(Y$,28)
 3630   *  B$=MID$(Y$,29,28)
 3635   *  C$=MID$(Y$,57,28)
 3640   *  D$=MID$(Y$,85,13)
 3645   *  E$=MID$(Y$,98,2)
 3650   *  F$=MID$(Y$,100,9)
 3655   *  G$=MID$(Y$,109,2)
 3660   *  H$=MID$(Y$,111,3)
 3665   *  I$=MID$(Y$,114,7)
 3670   *  J$=MID$(Y$,121,5)
 3675 <*  RETURN
10000      !
10005      ! 05/10/81 Input
10010      !
10015 >*  PRINT O$(0)
10020   *  PRINT "ENTER '0' IF OLD FILE, '1' FOR NEW
           FILE: ";
10021   *> R$=FAA(1)
10022   *  IF R$<>"0" AND R$<>"1" THEN 10021
10023   *  PRINT R$
10030   *  I%=1:IF R$="0" GOSUB 30015:GOTO 10060
10035   *  INPUT "ENTER NAME OF NEW FILE.";R$
10040   *  R$="N:1:"+R$
```



```
10045   *  OPEN 1 R$
10055   *  GOTO 10080
10060   *> GET 1 RECORD SIZE(1) Z$
10065   *  IF MID$(Z$,126,1)=" " Z$=LEFT$(Z$,125):I%
           =2:PUTSEEK(1)=SIZE(1)
10080   *> A$="":B$="":C$="":D$="":E$="":F$="":G$=""
           :H$="":I$="":J$=""
10085   *  GOSUB 515
10090   *  IF LEN(A$)=0 THEN GOTO 10130
10095   *> GOSUB 1115: ! Display
10100   *  GOSUB 3015: ! Modify
10105   *  IF LEN(R$)>0 THEN GOTO 10095
10110   *  IF I%=1 Z$=A$+B$+C$+D$+E$+F$+G$+H$+I$+J$
10115   *  IF I%=2 Z$=Z$+A$+B$+C$+D$+E$+F$+G$+H$+I$+
           J$:PUT 1 Z$:Z$="":I%=0
10120   *  I%=I%+1
10125   *  GOTO 10080
10130   *> IF I%=2 Z$=Z$+REPEAT$(" ",125):PUT 1 Z$
10135 <*  RETURN
11000      !
11005      ! 05/10/81 Display to CRT
11010      !
11015 >*  PRINT O$(0)
11020   *  INPUT "ENTER STARTING FILE";C%
11025   *  GETSEEK(1)=INT((C%+1)/2)
11030   *  Q%=C%-(INT(C%/2)*2)
11035   *  IF Q%=0 Q%=2
11040   *> GET 1 Z$
11045   *  FOR P%=Q% TO 2
11050   *>      IF P%=1 Y$=LEFT$(Z$,125)
11055   *       IF P%=2 Y$=RIGHT$(Z$,125)
11057   *       GOSUB 3625: ! Separate fields
11060   *       GOSUB 1115: ! Display
11065   *>      S%=IN(F%)
11070   *       IF S%=32 THEN 11065
11075   *  NEXT P%
11080   *  IF RECGET(1)<RECPUT(1) THEN Q%=1:GOTO 110
           40
11085 <*  RETURN
11090      IF N<RECPUT(1) GOTO 11050
11100 <*  RETURN
12000      !
12005      ! 05/10/81 Modify
12010      !
12015 >*  N%=0
12020   *  PRINT O$(0)
12025   *> PRINT " Enter Record to be Modified, or (
           0) to Exit."
12027   *> R$=""
12030   *  INPUT R$
12035   *  IF LEN(R$)=0 PRINT "ERROR*****: YOU MUST
           ENTER RECORD NUMBER, OR A ZERO!": GOTO 12
           027
12040   *  FOR J%=1 TO LEN(R$)
12045   *       IF MID$(R$,J%,1)<"0" OR MID$(R$,J%,1)
                >"9" PRINT "ERROR*****: ENTRY MUST BE
                NUMERIC!":GOTO 12027
12047   *  NEXT J%
12050   *  C%=VAL(R$)
12055   *  IF C%=0 GOTO 12135
12060   *  G%=INT((C%+1)/2)
12065   *  P%=C%-(INT(C%/2)*2)
12070   *  IF P%=0 P%=2
12075   *  GET 1 RECORD G% Z$
12080   *  IF P%=1 Y$=LEFT$(Z$,125)
12085   *  IF P%=2 Y$=RIGHT$(Z$,125)
12087   *  GOSUB 3625: ! Separate Fields
12090   *> GOSUB 1115: ! Display
12095   *  GOSUB 3015: ! Modify
12100   *  IF LEN(R$)>0 THEN GOTO 12090
12105   *  IF P%=1 Z$=A$+B$+C$+D$+E$+F$+G$+H$+I$+J$+
           RIGHT$(Z$,125)
12110   *  IF P%=2 Z$=LEFT$(Z$,125)+A$+B$+C$+D$+E$+F
           $+G$+H$+I$+J$
12115   *  PUT 1 RECORD G% Z$
12120   *  GOTO 12025
12135   *> PRINT "TOTAL LINES MODIFIED =";N%
12150 <*  RETURN
13000      !
13005      ! 05/10/81 Display Last Record
13010      !
13015 >*  PRINT O$(0)
13020   *  GET 1 RECORD SIZE(1) Z$
13025   *  PRINT Z$
13030   *  PRINT
13035   *  PRINT "PHYSICAL RECORD =";SIZE(1)
13040   *  PRINT "LOGICAL RECORDS =";SIZE(1)*2-1;SIZ
           E(1)*2
13045   *  PRINT "Press RETURN to continue."
13050   *  INPUT R$
```

```
13055 <*  RETURN
30000     !
30005     ! 12/28/81 Subroutine for displaying cont
          ents of disk;
30006     !                          - selecting file
          for opeation.
30010     !
30015 >*> PRINT O$(0)
30019  *  OPEN 8 O$(3)+"DIR" ERROR 30075
30023  *  CLOSE 8
30027  *  PRINT TAB(10);"THE FOLLOWING FILES ARE AV
          AILABLE:"
30031  *  PRINT
30035  *  DISPLAY O$(3)+"DIR"
30039  *  PRINT
30043  *  PRINT "If desired file is not listed, ins
          ert"
30047  *  PRINT "another disk, type 'X', press RETU
          RN."
30051  *  PRINT
30055  *  INPUT "Otherwise, Enter name of file desi
          red:";F$
30059  *  IF F$="X" OR F$="x" THEN 30015
30067  *  OPEN 1 O$(3)+ F$ ERROR 30075
30071 <*  RETURN
30075   > PRINT
30079     PRINT "*****";ERR$;"*****"
30083     PRINT "Correct Problem, Press RETURN to C
          ontinue."
30087     PRINT
30091     INPUT R$
30095     GOTO 30015
31100     !
31110     ! Configure
31115     !
31120 >*  DIM A$(28),B$(28),C$(28),D$(13),E$(2),F$(
          9),G$(2),H$(3),I$(7),J$(5),Y$(125),Z$(250
          )
31125  *  L$="  "+REPEAT$("-",28):M$=REPEAT$(" ",28
          )
31130 <*  RETURN
40000     !
40010     ! Main Menu
40020     !
40060 >*> PRINT O$(0)
40070  *  PRINT "**********INPUT & MODIFY ROUTINES*
          *********"
40080  *  PRINT
40100  *  PRINT "0 - EXIT INPUT & MODIFY ROUTINES"
40110  *  PRINT "1 - INPUT RECORDS"
40120  *  PRINT "2 - DUMP RECORDS TO CRT"
40130  *  PRINT "3 - MODIFY RECORDS"
40140  *  PRINT "4 - DISPLAY FINAL RECORD"
40170  *  PRINT
40180  *  PRINT "Enter number of function desired:
          ";
40181  *> K$=FAA(1)
40182  *  IF K$<"0" OR K$>"4" THEN 40181
40183  *  PRINT K$
40190  *  IF K$="0" THEN GOTO 40335
40210  *  IF K$="1" GOSUB 10015:GOTO 40310
```



```
40215  *  GOSUB 30015
40220  *  IF K$="2" GOSUB 11015:GOTO 40310
40230  *  IF K$="3" GOSUB 12015:GOTO 40310
40240  *  IF K$="4" GOSUB 13015:GOTO 40310
40310  *> CLOSE 1
40320  *  INPUT "Press RETURN to continue.";E$
40330  *  GOTO 40060
40335 <*> RETURN
57100     !
57105     ! 12/20/81 Read Clear Screen
57110     !
57115 >*  O$(0)=""
57120  *  FOR N%=0 TO 2
57125  *       O$(0)=O$(0)+CHAR$(PEEK(J%+N%))
57130  *  NEXT N%
57135  *  O$(2)=CHAR$(PEEK(J%+N%+1))+":"
57140  *  O$(3)=CHAR$(PEEK(J%+N%+2))+":"
57145 <*  RETURN
57200     !
57205     ! 12/20/81 Test for Rev. 3/4, DIM O$
57210     !
57215 >*  J%=16R2F06
57220  *  IF PEEK(16R04C9)=64 THEN J%=16R2F7A
57225  *  IF PEEK(16R04C9)=64 THEN DEF FAA=16R2F80
57230  *  IF PEEK(16R04C9)=0 THEN DEF FAA=16R2F0C
57235  *  DIM O$(3,8)
57240 <*  RETURN
Variable sizes:    Real - 5
Integer - 3     String - 40
Logical memory end:  DFFF H
0000 H bytes are reserved in high memory
FA*   A

FN*

N
C% F% G% I% J% N% P% Q% S%
A$(28) B$(28) C$(28) D$(13) E$(2) F$(9) G$(2)
H$(3) I$(7) J$(5) K$(40) L$(40) M$(40) R$(40)
S$(40) T$(40) Y$(125) Z$(250)


O$(3,8)


Memory Allocation
====== ==========

Interpreter:     22272  =  5700 H
Program:          6022  =  1786 H

Real Var:          130  =  0082 H
Integers:           78  =  004E H
Strings:           776  =  0308 H
Arrays:             44  =  002C H
Total Var:        1028  =  0404 H

Total Alloc:     29322  =  728A H
Available:       57344  =  E000 H

Dynamic allocation & buffers not included
```

MUG SUPPLIES & COMMENTS

If you wish to comment on the MUG newsletter, or have a question, a classified ad to place, or wish to purchase any of the supplies in the adjacent column - jot a note below, check any purchase, cut off the bottom of this page and send to MUG, 604 Springwood Circle, Huntsville AL 35803, USA.

1. ( ) $ Free Commercial MDOS S/W Price List
2. ( ) $ Free Commercial CP/M S/W Price List
3. ( ) $ 1.00 Commercial MDOS S/W Catalog
4. ( ) $ 2.00 Commercial CP/M S/W Catalog
5. ( ) $ 1.00 MUG MDOS Library Catalog
6. ( ) $ .50 MUG CP/M Library Catalog
7. ( ) $42.00 Scotch DD, Reinforced Hub Disks(10)
8. ( ) $18.00 Year 1 MUG Back Issues(12)
9. ( ) $18.00 Year 2 MUG Back Issues(12)
10. ( ) $ 2.00 Micropolis Drive Head Pads(2)
11. ( ) $ 5.00 Lube, for Micropolis Drives(tube)
12. ( ) $71.00 Micropolis (MDOS) Ver. 4.0
    Incls. Manual, BASIC, and all Utils
13. ( ) $50.00 MDOS Ver. 4.0 Manual only
14. ( ) $50.00 Micropolis Drive Maintenance Manual

Above items are postpaid to North America. Elsewhere, (airmailed) add $7.00 (each) for MDOS, Manuals, and back issues; add $1.00 (each) for Catalogs, lubricant, and pads. VISA & Master Card accepted. U.S. funds only.

AFORTH UPDATES

LAST CHANCE FOR BARGAIN PRICE

George Shaw tells me that the new FORTH will be out on the first of August. The tentative price is $325-$375. For those of you interested, any order for the current AFORTH (at $150) before August 1st will get you the update free. See page 2 of the March newsletter for a description of the software, and the offer. Outside North America, add $7 for air-mail delivery.

AFORTH PATCHES

A few problems have arisen, particularly for the EXIDY, with AFORTH operation. They may be bothering other users, too. The new version will have a configuration routine built in, so these problems will not occur.

To change the Clear Page, or Formfeed, control, type in:

```
xxx ' PAGE 2+ !
```

where xxx is the decimal character that clears screen on your terminal.

To change the input for backspace, that is, the keyboard character pressed, type in:

```
xxx 14 +ORIGIN !
```

To change the output for backspace, that is, the monitor response character, type in:

```
xxx 54 ' EXPECT + !
```

Again, xxx is the decimal representation of the character which performs the funtion on your equipment.

These patches could also be added to one of the Editor screens so you wouldn't have to type them in each time.

To permanently save them in the executive AFORTH, perform the following:


```
HEX
HERE.
```

(FORTH will give you a number back.)
(Go back to MDOS. Scratch AFORTH - from a copied disk, in case this doesn't work.)

```
SAVE 2B00 here 18 3EFE
```

where 'here' is the number that was returned.

One other problem that isn't as simple to patch, is the graphics coming out on the last character of the AFORTH text strings, on the EXIDY. The problem is because neither the EXIDY or the MDOS executives is zeroing the high-order bit before sending it to the screen. To fix this item, you must patch MDOS to do an ANI 7F somewhere in the output routines. This will be cured in the updated version, also.

.....

CLASSIFIED

FOR SALE: IDS 225 printer w/graphics. Good condition. Works fine. $300 shipped prepaid in US.

Al Seyle, 2218 Via Tomas, Camarillo CA 93010
(213) 889-5400 7AM-6PM PDT M-Th, (805) 987-1947 any other time.

.....

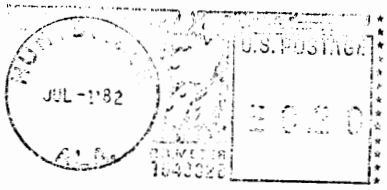
Published Monthly by the MUG
Subscription rates:
U.S., Canada, Mexico; $18/year: Other, $25/year

FIRST CLASS MAIL
===== ===== ====

FIRST CLASS MAIL
===== ===== ====

MICROPOLIS USERS GROUP

Buzz Rudow, Editor
604 Springwood Circle
Huntsville AL 35803
(205) 883-2621



FIRST CLASS MAIL
===== ===== ====