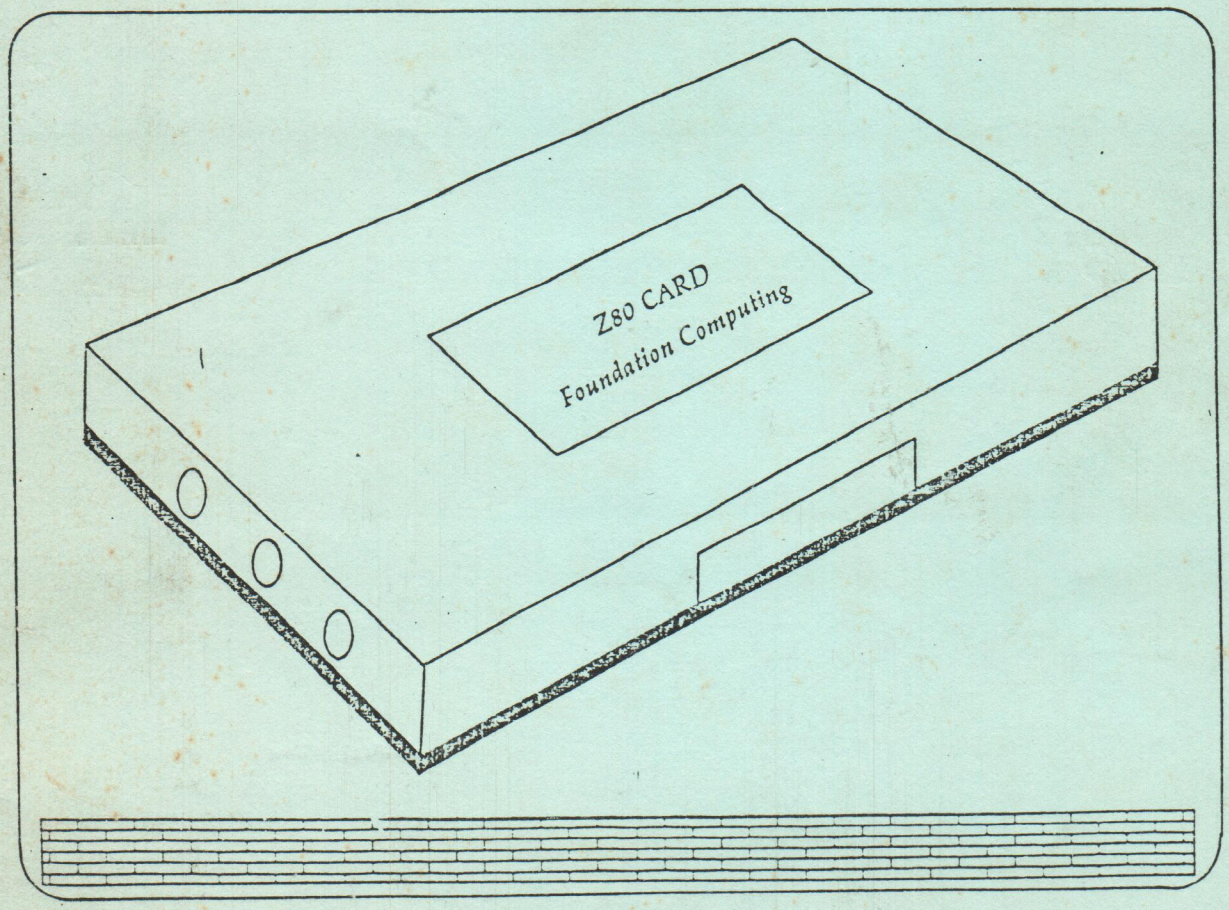
Foundation Computing

Z80 CARD



This document describes the Foundation Z80 CARD.

"Read Me" File

Ignore the instructions in the Users' Guide that ask you to type out the file "READ.ME". This sheet is being enclosed instead.

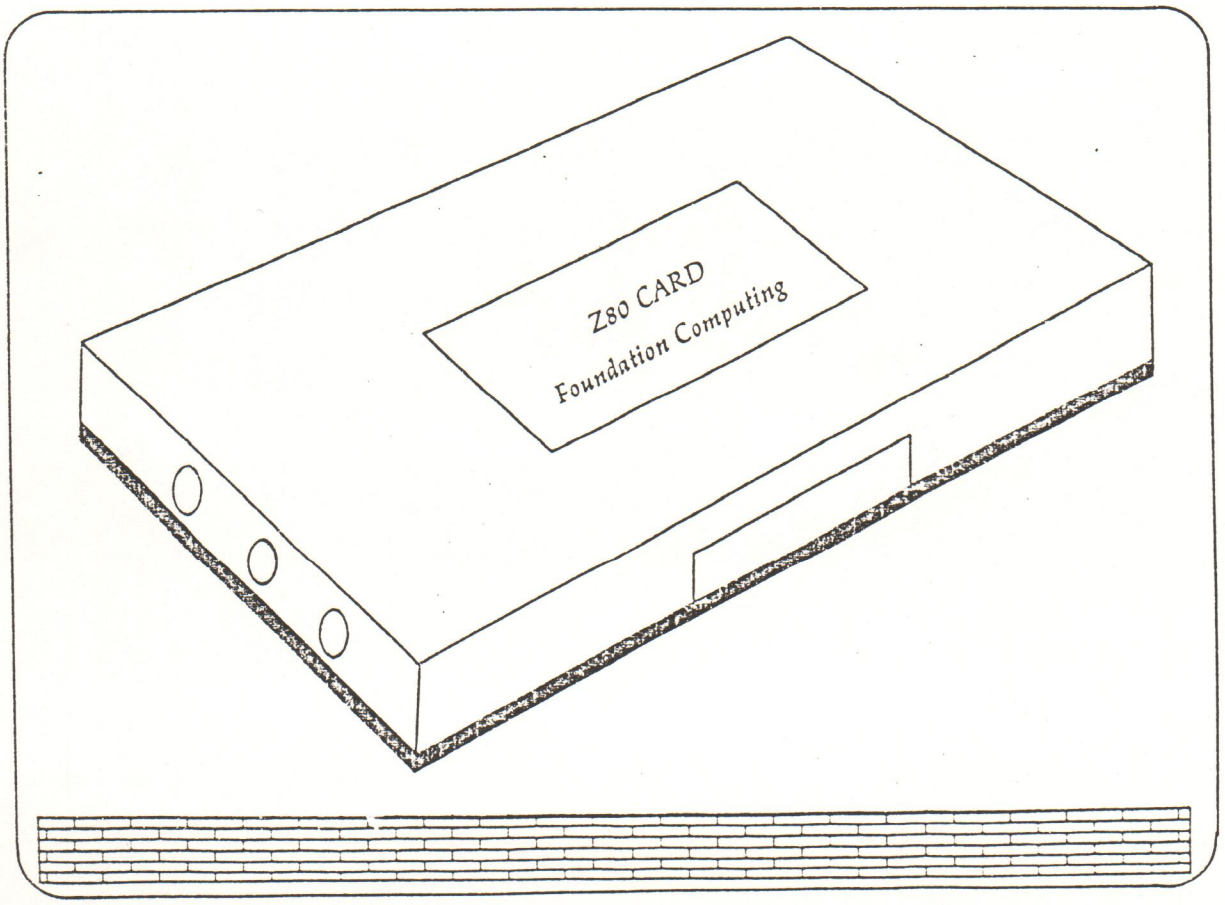
After reading the "Getting Started" section in the manual, please note the following: you may no longer need the "terminating resistor pack" in your disk drive after you install your Z80A card. Initially, install your Z80A card and leave the terminating resistor pack installed wherever it is presently located, but if you encounter disk errors then either remove it or reinstall it on your internal disk drive.

Another note about disk drive installation: the piggyback disk adaptor cable plugs in with the blue edge stripe on the cable pointing upward. (If you want to double check, this means that the uppermost contacts on the slotted blue connectors are numbered "1" and "2".) Hold the cable in front of you so that all connectors open toward you - that is, on the three blue connectors, the slots face you, and on the black connector, the holes face you. The black connector should be on the far left. Now stretch the black connector out flat so that it extends beyond the left end of the cable. If you face your peripheral expansion box, this is the way that the cable is designed to plug into your disk system. In the PEB, your disk drive will be on the far right, the TI disk controller will be next to it, and the Z80A card will be on the left of the TI disk controller. You do not need to twist the Piggyback Disk cable to plug it into the back of the TI disk controller, and the right-hand part of the cable that goes to your external drives should only need to be twisted if your present cables are twisted. One final note about which slot to install the Z80A card into: you can either install it next to your disk controller card or one slot over. If you do the latter, it's easier to press the reset button.

After you install the Z80A card, check out your TI system for basic sanity before putting your only copy of a master diskette into a disk drive. Take a diskette that you can afford to lose - a blank one for example - and make sure that you can read and write from your disks using the 99/4A by itself. If wires get bent or twisted in a cable, your disk drives can blithely write over a diskette even though it has a write-protect tab on it.

Foundation Computing

Z80 CARD



This document describes the Foundation Z80 CARD.

Section 1 - Introduction

Welcome aboard! This manual is for owners of the Foundation Z80A Card, a complete computer system on a single card that plugs into the Peripheral Expansion Box of a Texas Instruments 99/4A computer. We know from talking to you all on the phone that you span a tremendous range of knowledge and interest. This manual attempts to be all things to all levels of knowledge. In so doing, it runs a risk of being too complicated for a novice or too simplistic for an expert. When we've had to make choices, it's leaned in the direction of being too technical because you can probably find a friend that can read the description and explain it to you. Besides, you'll probably want the answer to some complicated questions yourself during your (hopefully) long ownership of this product.

There are six sections to this manual, organized as follows:

1. Introduction. This is where we are now. This section gives an overview of the manual.

2. Getting Started. When I get a new piece of equipment, I want to plug it in immediately and make sure that it survived being shipped. This section assumes that you feel the same way, and assumes that you probably haven't even taken the time to read the rest of the manual.

3. The Z80A in the TI 99/4A World. A quick look at how the Z80A card fits in with the rest of your TI 99/4A system. With two complete computer systems operating at once,it can be a little complicated figuring out which one is doing what. This section tries to leave you with some understanding of how everything fits together.

4. Where to Get More Information. In order to do anything useful with a computer, you have to have some "application programs" to run on it. There is a whole world of CP/M - compatible software, and we'll try to get you started on finding the software packages you need. Also, the Z80A chip has been around for a long time, and there are probably a dozen places to turn for any piece of hardware information you want. We'll mention a few of our favorite sources.

5. Introduction to the MRS Operating System. If you've already started looking at the MRS Operating System manual, you're either feeling delighted or overwhelmed. The good news is, unless you're writing programs, you'll only need a cursory overview of what the operating system does. We are providing the full, detailed technical documentation packet because 1) some of you are programmers and will need it very much, and 2) just as with the owners' manual for a car, you may eventually need to look up some detail even though you

don't need to know it in your day to day driving. Anyway, this section tells you a little bit about what operating systems do and and gets you started on understanding the MRS technical documentation.

6. The Z80A Hardware. This is for those of you that want to go deep into the innards of the Z80A card, perhaps writing device drivers for laboratory experiments or interfacing a laser printer directly to one of the parallel ports. You get the idea... this is the "Extra for Experts" section.

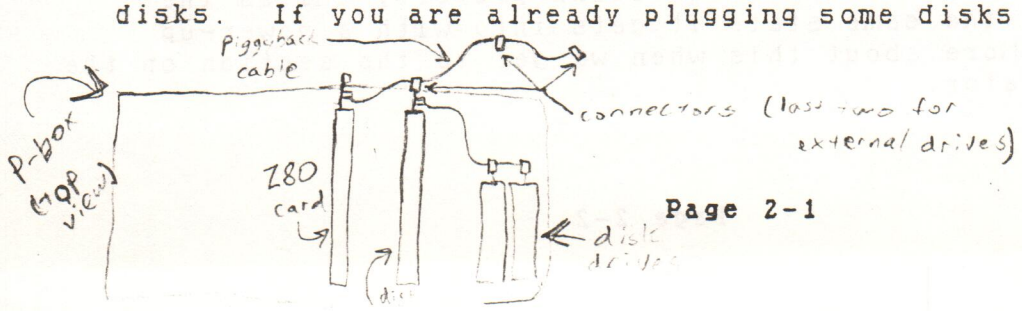
You've already installed a number of other cards into your TI 99/4A expansion box, so let's skip all the details of how you insert the card into the box. The one thing that deserves repeating is that you really should wait a full two minutes after turning the power off before you start plugging or unplugging cards. Enough said about things you already know.

First, unwrap the Foundation Z80A card and take a close look at it, especially at the back. Notice that the part of the card that is outside the case looks like a cross between an RS232 card and a disk controller card. That's because it duplicates the function of those cards. The 25 pin plastic DB25 connector at the very back is wired just like a TI RS232 card, and the 34-pin shielded header that is next to it is wired just like TI's disk controller.

Let's discuss the disk drive connection first. You already have your disk drive(s) connected to your TI disk controller through cables of some sort. The Z80A card needs to be hooked up in parallel to these same disk drives. When you get finished with the hookup, either the TI or the Z80A will be able to use any drive; naturally, when the TI is using a drive, it needs to have a TI-formatted diskette in it, and when the Z80A is using a drive, it needs to have a CP/M-formatted diskette in it.

Your disk drives are already hooked up in parallel to each other. That is, each wire from the TI disk controller runs to the same pin of each disk drive's edge connector. This is sometimes called a 'daisy chain'. Now you need to connect the Z80A into this same daisy chain, just as if it were another disk. If all your TI disks connect to the side of your TI disk controller, then you simply run the piggyback disk cable from the side of the Z80A card to the back of the TI disk controller (i.e., to the disk controller's 34-pin edge connector), and leave the two extra Piggyback connectors hanging. (Notice that in order to do this, the Z80A card must be located next to the TI disk-controller card.) The reason that this works is that the side and back ports of the TI disk controller are directly connected to one another - wires on the disk controller board directly connect corresponding pins.

If you have one or two internal drives, you probably have exactly the situation described above. Both drives are probably connected to the side port of the disk controller, and all you need to do to connect the Z80A disk controller is to plug the Piggyback Disk cable into the side port of the Z80A and plug one of its connectors into the back port of the TI disk controller. Things are only a little more complicated if you have external disks. If you are already plugging some disks into the back of



the TI disk controller, then you need to 1) disconnect them and set aside their cables, 2) wire the Z80A to the back of the TI disk controller as above, and 3) plug the two extra edge connectors on the Piggyback cable onto the edge connectors of your external disk drives. (Depending on who manufactured it, you may have to unscrew your disk-drive case to get access to the disk-drive edge connector.)

Since you already had your disks working with the TI system by itself, you shouldn't have to do any cutting of drive-select shunt packs or modification of terminator resistor location. Nevertheless, recall that you should only have one terminating resistor pack in the daisy chain, and it should supposedly be located in the farthest disk drive from the controller. (Actually, I've seen "plain vanilla" TI systems that got consistent errors with the terminating resistors located 'correctly', but that worked fine with the resistor pack mounted in the internal drive. You might want to experiment with this if you are getting disk errors.) Also, each disk drive has a 'drive-select' jumper that physically identifies the drive number at a hardware level. You probably only need to think about these things again if you are changing your disk-drive configuration while you are installing your Z80A card. Better yet, why not just change one thing at a time?

Now for the RS232 connection. The RS232 connector on the Z80A card is wired to correspond to the TI RS232 connector. This has two useful side effects. One is that the 'Y-adaptor' cables that TI manufactured will work. They are available from several dealers for very reasonable prices. The other is that for directly connecting the Z80A card to a TI RS232 card, a direct, straight-through 25 pin connection will wire everything correctly. The sample terminal emulator program that is provided with the Z80A card is set up to make serial port 1 on the TI 99/4A ("RS232/1.") work as a terminal to the Z80A card. So, unless you modify the terminal emulator program (more about this later), you should either wire all ports directly across, which is easiest to do using a flat ribbon cable with DB25 plugs on each end, or at least wire port 1 directly across. At least the following pins need to be connected for port 1: DB25 pins 1,2,3,5,7,20.

<u>Booting Up for the First Time</u>. That wasn't too bad, was it? Now you are ready to apply power for the first time. Check that all cards feel snug in the expansion box, then turn it on and power up the console. Don't forget to turn on the console! With the Z80A connected in parallel with the TI disk controller, the Z80A card will not be able to access disks properly unless the console is in the same state it gets into with a power-up initialize. More about this when we get to the section on the terminal emulator.

At this point, the TI should be acting just as it always has. That is, you should see the color bar screen first, then you should be able to run programs from any disk drive just as you always have. Take a moment to verify this. What you don't see yet is that the Z80A is also running. The only evidence you have that anything has changed is that the pilot light is on in the Z80A card's slot. The Z80A card expects to interact with a terminal connected to its first RS232 port, and you haven't yet told the TI 99/4A that you want it to behave like a terminal.

That is the next step. There are three diskettes distributed with the Z80A card. Insert the one labeled "Terminal Emulator" into DSK1. If you are using the Editor/Assembler module to load the terminal emulator, select Option 3, "Load and Run", then load and run a file named P99. From Extended Basic, type

RUN "DSK1.ZE2" ("DSK1.ZE3" for enhanced version) loading enhanced version from E/A, 1) select Option 3 2) File Name: DSK1.FTERM 3) Program Name: TERM

The disk should activate momentarily, then the screen should display a signon message. The 99/4A is now behaving as a terminal to the Z80A card, and when you type <ENTER> you should see a bracket ">" prompt coming from the Z80A power-up monitor. (It may be preceded by an error message if your RS232 port had some noise on the line before you started the terminal emulator.)

Before going on, let's take a few minutes to look at some of the monitor commands that are available. You won't use most of them very often unless you are doing assembly-language program development, but it's interesting to know that they're there. Type "MU" to get a menu of the monitor commands that are available. The monitor is a low-level diagnostic tool that lets you examine what the Z80A hardware is doing. The commands are self explanatory if you program in assembly language; for example, the menu indicates that the command to read and display the contents of a memory area is "DM <ha> <hc>" , where <ha> stands for an address in hexadecimal and <hc> stands for a count of the number of bytes to display, also in hexadecimal.

The real reason I asked you to type the MU command was that it generates a fairly lengthy amount of text, and I wanted to point out a few things about this version of the terminal emulator software. We initially tried a version of a terminal emulator that did traditional, line-at-a-time vertical scrolling, and were disappointed to find that the 99/4A processor performance held us back to a slow display. We speeded things up considerably by eliminating traditional scrolling and going from the end of one page directly to the beginning of the next. Now, the purpose of scrolling is so that you can see the most recent screen's worth of information at any time. We decided to take

advantage of the considerable amount of memory available to go this one better: we saved the last fourteen pages of memory in a wraparound buffer which you can look at by typing <FCTN> 6 , "Roll Down" to start things off, then <FCTN> 6 or <FCTN> 4, "Roll Up", to page around in the buffer. (The screen color is altered to let you know that you're in this mode.) Typing any other key will take you out of review mode and resume sending your keystrokes to the Z80A. One other thing to note is that when you get to the end of the fourteen page buffer, you should type a <FCTN> 9 "Escape" to clear out the entire buffer, otherwise you may lose characters at the end of each page while the terminal emulator pauses to clear out the next page - again, the terminal emulator notifies you when this is due by changing the screen color. (These color changes also show up on a monochrome screen, of course, though not as vividly.) Before you do much more, it would be a good idea to use the TI Disk Manager command module to back up your terminal emulator disk.

Now load the MRS operating system diskette. Drive 1 on your system must be a forty track drive. If it is a single-sided disk drive, select the MRS diskette labeled "1S2D"; if drive 1 is double sided, select the diskette labeled "2S2D". Insert the diskette in drive 1, type "BT<ENTER>" (leave the alpha lock key down while typing commands to MRS), and see if the MRS operating system loads.

Did you get the message, "Welcome to Foundation's Z80A Card"? If you did, congratulations! Another delicate electronic product has survived the tender mercies of UPS shipping. You are ready to begin running CP/M-compatible software. If not, check that you are typing in upper case letters, then check the section on troubleshooting problems.

★ Before Booting the system type the CD command (Configure Drives) and respond to the menu prompts for the format of Disks you'll be using, i.e. 40 track single side "KAYPRO 2" format

There is one piece of unfinished business to tend to first. Before you do anything else, you should back up your system diskette. Here is a cookbook procedure to do this.

1) Format a blank diskette. Starting with the MRS operating system prompt (e.g. "A."), type "FORMAT". Specify the disk drive you want to format the new diskette on. (Under CP/M-like operating systems, disk drives are referred to as A:, B:, and C: instead of DSK1. , DSK2. , and DSK3.) Specify the disk format you want, which for most of you will be either SSDD (Single Sided Double Density) 40 track ("Kaypro 2" format), or DSDD 40 track if you have a double sided drive. Remember that when the display flashes past you on the screen, you can reexamine it at your leisure by typing <FCTN> 6 and <FCTN> 4 to roll backwards and forwards through the buffer. DO, please, make sure that you have replaced your system diskette with a blank diskette before you tell MRS to begin formatting it. Formatting destroys all the stored information on a diskette, and if the diskette you destroy is your only copy of the master, ... well, life is like that sometimes! Anyway, the system will format the diskette, then you can type 4 to return back to MRS.

2) Replace the MRS system diskette in drive A, if you removed it to format a new diskette. Now back up the MRS system diskette onto the newly formatted diskette:

A) If you have only one disk drive, type "SSP", then in response to the "SSP.A>" prompt, type "COPY A:" This does a full disk to disk copy of a double density diskette, so it takes four passes to copy a single sided diskette or eight to copy a double sided diskette. You must keep track of how many passes have occurred; the COPY utility will not terminate when it has gone too far.

B) If you have more than one disk drive, you can speed this operation up tremendously by copying from one diskette to another. If both the source and destination disk drives are the same type, then you can use the COPY command as follows: "COPY destination,source", for example,

COPY B:,A:

where both B: and A: are SSDD 40 track drives. Alternatively, you can use the XFER command as follows to transfer all files from A: to B:

XFER B:=A:*.* ← use this instead of COPY.

(COPY works, but it keeps making disk passes ad infinitum even after the copy is complete, and you have to reset the system manually with the reset button on the back of the card)

This will work across different types of disk drives. If you used the XFER command, then when the file transfer is finished type

INIT B:,MRS

The COPY, XFER, and INIT commands are fully covered in the MRS technical manual, and you should read those sections of the manual for more information.

NOTE: Two things are important to know about when getting started on the system, though they don't really fit into the flow of this discussion. One of them is mentioned in the MRS manual but it should probably be written in large, red letters: whenever you change diskettes on CP/M-like operating systems, you have to let the operating system know about it! You do this by typing a ^C (control C) from the operating system prompt or from the SSP prompt. You'll probably lose a file or two until this becomes second nature. The other thing is the reset button. When you were examining your card, you probably noticed a little pushbutton switch on the back of the card below the RS232 connector. This is the reset switch, and it will take the Z80A card back to the state it is in on power-up. You tend to use the reset button fairly often under CP/M, and we deliberately selected a switch that would be easy to mount off the board. That is, if your computer tends to stay in one place, you may want to connect longer wires to the reset button and mount it somewhere closer at hand. Just try to avoid pushing it while the disk drive is active! If a read or write is taking place, you're almost guaranteed to lose some data.

3) At this point, you have a duplicate disk to work from. Label it, and put the master away in a safe place. Now, type ^C to let the operating system know you're using the current disk, and run the SSP program. From within the SSP program, type

TYPE READ.ME

and see any last minute instructions or corrections that were distributed with the software package. The READ.ME file will also describe any demonstration programs that were included with this release of the product.

Section 3 - The Z80A in the TI 99/4A World

As stated in the introduction, this chapter takes an introductory look at the hardware of the Z80A card with a special perspective. The Z80A is very nearly a self-contained computer, but you typically interact with it using the rest of your TI 99/4A system as a console. This leads to a situation that usually only occurs at wild California parties, where you stare around yourself in amazement trying to figure out who is who is doing what to whom. With a little luck, we can clarify that situation now.

First of all, the Z80A card doesn't especially know or care that it is running in conjunction with a TI 99/4A. All it knows is that somebody is using a terminal to interact with it through its console port. Now, the cheapest possible way to connect a terminal to the Z80A card is to use your console as a terminal. This means that you have to run some piece of software that makes the TI 99/4A behave like a terminal. That is, the 99 needs to keep an eye on the keyboard and send any characters that you type over an RS232 wire to the Z80A. Conversely, the 99/4A needs to keep an eye on the incoming characters that the Z80A sends, and display them on the screen. In brief, that's all that a terminal emulator does. You could probably write your own in a matter of hours. At least, that's what we thought before going into it. It turns out that there are some tricky aspects to synchronizing the two machines that need to managed rather carefully; the 99/4 can be amazingly, frustratingly slow at some tasks! The present version of the terminal emulator is directly adapted from something that Randy Holcomb contributed to the public domain. (See Computer Shopper, September and October '84 issues for more information. Also, Randy is very actively involved with the CompuServe Texas Instruments SIG, as you know if you subscribe to CompuServe.) Randy did us all the great favor of unearthing some fundamental information about how you use the RS232 capabilities of the 99/4A. We've extended his contribution with some interesting capabilities for saving previous screens, and, especially, with the capability of working with the Z80A card at 9600 baud..

CP/M was written to be a very general-purpose piece of software that mediates between an application program and whatever hardware it happens to be running on. Its goal is to act as a common denominator that lets the software be as independent of a particular computer system as possible. With this in mind, let's take a look at just what kind of computer system this CP/M-compatible operating system is running on.

First of all, let's look at the heart of the system, the microprocessor chip and its accompanying memory. The Foundation Z80A card obviously uses a Z80A. This means that all CP/M-80 software will run on it. Some other CP/M systems use Intel 8080 or 8085 chips; the more limited instruction set of these chips prevents you from running some important programs. Accompanying this CPU chip is 64 Kb of high-speed dynamic memory, driven by a special Texas Instruments memory-controller chip. 64Kb is the

maximum amount of memory that the Z80A chip can directly address without resorting to subterfuges like bank switching. What this means in practical terms is that essentially all CP/M-compatible programs will run on the machine; a very few CP/M programs have been written that make use of bank-switching and the like so that they will only run on one particular machine, but usually if you hear about CP/M machines with more than 64Kb of memory, the remainder of the memory is being used as a RAM disk or to set up several 64Kb programs so that you can quickly switch from one to the next.

When the Z80A card is initialized, i.e. when you power up the expansion box or push the reset button, it begins executing instructions out of startup ROM that is located at address 0. These instructions copy two pieces of code from ROM up into RAM memory: a startup monitor and the BIOS ("Basic I/O Control System") for the operating system. At the present time, this code takes up 8Kb of the 16Kb that could be stored in ROM, so it is possible that in the future Foundation will release other ROM-based software packages. If this happens, they will be installed by just opening the Z80A card case, unplugging one chip from a socket, and plugging in its replacement.

The Z80A card has two chips on it that provide "ports" to the outside world; that is, communications channels through which information can be transferred. One of these chips is a Z80A DART chip - "DART" stands for "Dual Asynchronous Receiver Transmitter". (On some production runs of the Z80A card, a functionally equivalent Z80A SIO/0 chip has been substituted.) The DART provides two independent serial channels that can communicate with devices like terminals, modems, or serial printers. Both of these channels emerge from the card through one DB25 connector; the chapter on "The Z80A Hardware" gives information about pinouts and signal names. One serial channel is used by the terminal that you use to interact with the Z80A card: the "console" or "CON:" in CP/M parlance. The other one is nominally referred to as the printer port though it can be used instead for any other serial device. Given that you already have your printer connected to an RS232 card in your TI expansion box, one practical suggestion is that you leave your printer connected to your TI RS232 card and modify the terminal emulator software to divert printer output to the printer as well as to the video screen. This will help you to avoid swapping cables back and forth between the TI RS232 card and the Z80A RS232 port.

For other I/O needs, a Z80A Parallel I/O device ("PIO") has been built into the Z80A card. It provides two independent 8-bit bidirectional parallel channels with full handshaking capability. Rephrased into English, what this says is that there is the Z80A card includes a chip that provides two additional communications channels. Each channel allows one byte of data to be transferred to or from an external device - for example, a printer or a parallel port on another computer. These

transfers can take place at very high speeds, and there is hardware built into the chip to make sure that the sender doesn't send faster than the receiver is ready to receive. These parallel ports emerge from a zero-profile twenty pin socket mounted next to the "etch fingers" where the Z80A card plugs into the expansion box. You can open up the Z80A card case and see where the socket is located - it is labeled "J3" on the silkscreen. The section on "The Z80A Hardware" tells you everything you need to know to construct a cable for the parallel ports. You may also be interested in knowing that Foundation Computing will be offering a low-cost board that connects the Z80A card directly to the TI 99/4A at high speed through the parallel ports. We'll send you information about this in a separate New Product Announcement.

One other thing needs to be clarified, and that is the whole business of how both a Z80A system and a TI 99/4A system can share the use of disk drives. How does this work, and what are the caveats? First, it should be mentioned that this is optional behavior. The price of a floppy disk keeps dropping, and you may find that you want your Z80A card and your 99/4A disk controller to be hooked up to unrelated drives. That would allow you to have up to seven floppy disk drives on your system! This may sound unlikely, but since your Z80A card can use 80 track drives and your 99/4A can't ...

Anyway, getting back to how this all hangs together: floppy disks are connected to a disk controller using something that engineers refer to as an "open collector" circuit. This type of arrangement allows several different pieces of equipment to all share a cable for passing information around. The rule that decides who is in charge is the following: whenever any one piece of equipment tries to drive a signal "low", it wins the argument and the signal becomes low. The only way that you can get a signal high is if everybody cooperates and leaves it high. This is crystal clear to you if you have an electronics background. Everybody else is scratching their heads wondering if they really need to understand this stuff. Not all of it. You just need to understand this: the 99/4A disk controller and the Z80A disk controller can both happily share the same disk cables as long as both controllers leave all the output lines high when they're not in charge. The Z80A disk controller will do this when it is powered up. So will the TI disk controller AS LONG AS THE TI CONSOLE IS ON! At some time during your ownership of the Z80A card, you'll forget to turn on the TI console, try to boot the Z80A card, and all you'll get is ten half-hearted thumping noises until it finally comes back and tells you something like "Boot Error". (Of course, you can also get this error by putting the wrong diskette in the drive!)

Can both the TI and the Z80A run programs at the same time? Yes, as long as the programs don't leave the disk controllers trying to drive a disk output line low. Obviously, it can be done because you're using that capability every time you run the terminal emulator program. However, for most of you, it's a pretty iffy thing that will probably get you in trouble. If you really want to get involved in writing concurrent programs, one piece of advice is that you should imitate the direct sector read at the beginning of the terminal emulator program, which homes the disk and guarantees that the floppy disk DIRC line is not driven low.

There we go again, getting completely incoherent. Sorry!

Section 4 - Where to Get More Information

Coming from the TI 99/4A world into the CP/M world can be a little disconcerting. There is just so much information and software available that you don't know where to start or who to ask. This section is an attempt to point you in some promising directions. Hopefully, you will also have an ongoing dialogue with Foundation Computing so that we can answer your questions and pass on your discoveries to other Z80A card customers.

First, there is the question of CP/M software. Everyone knows that there is a lot of it, but it's not always easy to locate a particular program. The best approach that we've found is to acquire a CP/M software catalog and make it the starting point for any search. Several catalogs of CP/M software exist. One favorite is the CP/M Software Directory, published by:

R.R. Bowker Company
205 E. 42nd St
New York NY 10017

This massive tome is updated annually. The 1985 edition has 755 pages of fine print listing available software and such pertinent information as where to contact the manufacturer. As of this writing, there is a toll free 800 number that you can call to request a ten day free examination of this book: (800)521-8110. If you go into your local bookstore or library looking for a copy of it, the International Standard Book Number is 0-8352-1973-9.

Incidentally, in case it seems that some of this discussion is too laudatory, it should be mentioned that Foundation Computing has no business relationship with any of the resources cited here. Our sole purpose in sharing this information with you is to assist you in getting value out of our products.

You may also be interested in locating dealers who can provide you with software or who can refer you to other sources. The best way to locate mail order dealers is to read the magazines in which they advertise. Two excellent sources are Byte Magazine, widely available at newsstands, and Computer Shopper. Computer Shopper is less known; it is a monthly newspaper dedicated primarily to advertisements. Some "ultra discount" advertisers use it to distribute computer products at some of the lowest prices in the industry. You can order a copy of Computer Shopper from:

Patch Publishing Co., Inc.
407 S. Washington Ave.,
P.O. Box F
Titusville, FL 32781
(305) 269-3211

Another source of information and software is your local CP/M users group. Computer Shopper publishes a list of hundreds of users groups that is updated monthly; there is probably one located within driving distance of wherever you live. Some of the larger user groups have undertaken the task of collecting and distributing public domain software that runs under CP/M. For example, the SIG/M Users Group library presently consists of more than 220 diskette volumes. Most local user groups have collections available for duplication; if you are unable to link up with someone that has a program you want, you may be able to locate a copy through the

Sig/M Users Group
P.O. Box 97
Iselin, NJ 08830-0097

Along these lines, one of the best sources for information and software is right in your living room! All of the public domain software we just mentioned and much good advice is available by dialing into a computerized bulletin board system. Most BBS's are free except for the cost of the phone call. In another league (and more costly!), there are the multi-user computer networks like CompuServe and The Source. If you're new to the whole world of telecommunication, you might pick up a copy of Mike Cane's book, The Computer Phone Book, published by

The New American Library Inc.
1633 Broadway
New York, NY 10019

Monthly updates to this book are available from

Mike Cane
The Computer Phone Book
175 Fifth Avenue, Suite 3371
New York, NY 10010

Finally, there is the particular software category of programming languages and tools. We feel that we have to acknowledge a few vendors that provide outstanding products at very reasonable prices. Most well known of these is Borland International, publishers of Turbo Pascal. If you purchase from them directly, the mailing address is :

Borland International
4113 Scotts Valley Drive
Scotts Valley, CA 95066

The Q/C package for the programming language C also provides an incredible value. For somewhere in the neighborhood of $100, The Code Works will sell you not only a well done implementation of the language C, but also the source code to the compiler and

all its libraries. They also have several other packages of interest to programmers, including a minimal $35 Z80 assembler, an editor, and a larger assembler/linker package that sells complete with source code for about $75. They are at:

The Code Works
Box 6905
Santa Barbara, CA 93160

Finally, the assembler/linker that we have used for much of our development work comes from 2500 AD Software, a company that specializes in cross-assemblers for just about every processor you can imagine. They are located at:

2500 AD Software Inc.
17200 E. Ohio Drive
Aurora, CO 80017
(303) 752-4382

Again, please let us know what your experience is with various suppliers. We will attempt to pass it on to other users of the Foundation Z80A card.

Section 5 - Introduction to the MRS Operating System

The manual accompanying this one is the guide to the MRS operating system. Not to worry. As far as a computer user is concerned, she may need to know about her machine at three levels. If she is really trying to stretch the machine beyond its normal limits, then the hardware is of concern. At this level, most of the manufacturer's documentation is useless. What you do is tape the I/O port addresses on the wall beside your terminal and keep the Z80A Technical Reference Manual from Zilog somewhere within arms reach. At the next level, there is the person who is trying to create software to run on the machine. At this level, the hardware is of some importance, but you usually interact with it one level removed - you let the operating system handle the grim details of how you initialize the floppy disk drive and you just write a program that reads or writes from files. At the third level - the level that most of us operate at most of the time - the computer and its system software are just a black box that let us run some piece of application software. At this level, the manual that tells you what you need to know is the manual for your applications software package - that is, the manual for your word processor, for your Basic interpreter, or for your spreadsheet.

The MRS User Manual describes the operating system in enough detail that you can write system calls to it if you are writing assembly language software. There are also some sections that everyone should read eventually - especially those that deal with everyday actions like listing a disk directory or transferring a file. For most people, though, it goes into too much detail. This chapter is an attempt to break it to you gently, and let you know which parts you can safely ignore. It also presents a little background on what operating systems do for you, since the TI 99/4A splits traditional operating system functions into several pieces.

Let's imagine that you received your Z80A card with no software. I know, it doesn't come with a word processor, etc., but suppose that it didn't have the software that is built into its ROM or the operating system diskette. What could you do with it? Well, in a few months (maybe a year), with about $10,000 worth of development equipment, a reasonably clever person could start from scratch and get it to read and write files from disk and to be able to run useful application software off the shelf. That is, to do the things it can do for you right now. It's all just a matter of juggling a lot of complicated details. That's what an operating system does. An operating system is a program that manages the hardware of a computer system so that application programs don't have to concern themselves with all those details.

On big mainframe computers - the IBM megaliths that fill whole floors of buildings - operating systems have turned into amazingly complex programs. It often takes a whole "systems programming" department just to make the operating system do what

it is supposed to do. On microcomputers, things are much simpler, partly because the operating system attempts to do far less. Basically, what MRS and other CP/M-compatible operating systems provide is 1) a filing system that lets you store and retrieve information using disk drives; 2) some conventions about how application programs should talk to devices like terminals; 3) some utility programs that let you control your computer; and 4) a way of loading and running application programs.

Compared to the many tasks that a big-machine operating system performs, this is a pretty short list. Nevertheless, some jargon has grown up around CP/M-like operating systems. You may have already encountered these terms; in any case, we should introduce them so that you can confuse all the other TI owners at your next users group meeting. First, you may have heard the term "BIOS". BIOS stands for Basic Input/Output System. The BIOS is a collection of short machine-language programs that deal with interfacing devices to your computer at a very low level. For example, the BIOS includes a program that just takes one character and sends it out over an RS232 port. Texas Instruments did an interesting thing with the idea of a BIOS when they designed the 99/4A system. Rather than having a BIOS for the entire computer, they split all those device interfacing routines apart and built them into the hardware of the cards you plug into the expansion interface. In TI parlance, these single-device BIOS routines are called "Device Service Routines", or DSR's.

Usually, application programs do not call BIOS routines directly. Doing so makes a program somewhat machine dependent, which is to say, unable to run on other CP/M-compatible systems. The safest way to use the operating system is to make calls to the "Basic Disk Operating System", or BDOS for short. If you'll take a moment now to leaf through the MRS User Manual, you'll see that a large section of the manual is dedicated to describing the "MRS User Calls", which are the ways that an assembly language programmer asks the operating system to do things like creating files or reading records from a file. Chances are that even if you do write programs, many of them will not be in assembly language but rather in some higher level language like Basic or Pascal. In that case, the person that wrote the language processor you are using made things more palatable by packaging them up in terms of the language you are using. For example, operating system call 11 checks to see whether a character is available from the console. I won't bore you with the details. In Turbo Pascal, you instead write:

```
If KeyPressed Then TakeSomeAction
```

and the built-in function "KeyPressed" fills in the rest for you.

Another component of the operating system is the "Console Command Processor", affectionately known as the CCP. This little piece of code examines commands that you type, splits them apart

a little bit, and takes the appropriate action. For example, the way that you run a program under MRS is by typing its name. The CCP looks at the characters you typed, loads the program, and transfers control to it.

Now, one last acronym. The whole purpose of this exercise is to load useful programs into the computers' memory and run them. The area in memory that you load your application programs into is referred to as the transient program area, or (you guessed it!) the TPA.

Enough of alphabet soup. Here's a quick picture of where all these pieces sit in main memory when your Z80A card is running:

	Z80A Card Memory
	Top of Memory (FFFF)
BIOS	The BIOS contains routines equivalent to TI Device Service Routines.
BDOS	The BDOS contains the bulk of the operating system.
TPA	The Transient Program Area is the area available for use by application programs.
Reserved	The first 256 bytes of memory are reserved for use by the operating system.
	Bottom of Memory (0000)

Remember that we said a few paragraphs ago that one of the th ngs CP/M-like operating systems provide is a "filing system" for information? You are already familiar with this concept from your use of the TI disk and disk controller. On the TI 99/4A you identify a file by specifying two things: a device name like "DSK1." and an arbitrary ten character name like "SOMEFILE". Under MRS and CP/M, files are specified similarly. Disk devices are called "A:", "B:", "C:", and "D:", where "A:" corresponds to "DSK1." etc. File names have a little more structure to them. The first eight characters of a file name can be any combination of letters or numbers, but should not

include the following special characters:

; : [ ] < > , = . ? *

since these characters are used for other purposes and their use would confuse the operating system. Following this file name, you may have an optional <u>file type</u>, which consists of a period followed by from one to three other letters or numbers. The filename type is used to group files that are similar. For example, you might choose to label all temporary files with a file type of ".TMP" so that you know it's all right to erase them. Some programs make assumptions about default file types. For example, Turbo Pascal will assume that a program has an extension of ".PAS" unless you tell it otherwise.

Filename types become especially useful when you are using what is known as the "wildcard" facility. MRS provides a package of utility programs known as the SSP, which we will get around to discussing in a moment. Suppose that you had labeled all temporary files with a file type of ".TMP" as suggested above. Then you could use the SSP erase command, ERA, to erase them all at once with the command

<u>SSP.A ></u>ERA *.TMP

(In this section, prompts that the computer sends will be underlined. The remainder of the line is what you type back to the computer.) When the SSP sees an asterisk in a file name or type, it will perform the action you specify on every file that matches the remainder of the name. So as well as matching all files that have a given file type, as above, you could match all files that have the same name regardless of file type:

<u>SSP.A ></u>ERA APRIL.*

will erase everything to do with the month of April, including APRIL.DOC, APRIL.TMP, and especially APRIL.TAX .
You can also wildcard both the file name and type.

<u>SSP.A ></u>ERA *.*

will erase everything on your disk, which will probably make you very sad if that's not what you meant to do. As well as the asterisk, which matches a whole file name or type, a question mark matches a single character in a file name or type. For example,

<u>SSP.A ></u>ERA APRIL.T??

will erase APRIL.TMP and APRIL.TAX, but leave APRIL.DOC intact. Wildcards are a powerful and useful way to operate on groups of files. Think about what you would have to go through with the TI Disk Manager Module to do the same things!

The System Support Program

We've already been making use of the System Support Program, or SSP, without stopping to become take a look at it. If you pick up any of the dozens of books about CP/M, you can read descriptions of utility programs with names like "PIP" and "SYSGEN". Under MRS, the functions of these programs are combined into one program which you run by typing

A SSP

The SSP lets you do most of the things that the TI Disk Manager Module does and more. Most of the housekeeping involved in managing your computer is done using the SSP. The most commonly used commands enable you to list disk directories and to erase, rename, and copy files. There are also commands to display files to the screen in several formats, to patch disk files (similar in purpose to the Navarrone Industries "Disk Fixer" program for the 99/4A), and to protect/release files. The section of the MRS User Manual describing the SSP is required reading for everyone. Fortunately, it is fairly easy to grasp.

One parenthetical note: You will probably supplement this documentation with some commercially published books about CP/M. Most of what you will read in those books applies equally well to MRS and other CP/M-compatible operating systems. The one area where this is not true is the SSP. The SSP performs the same functions as various CP/M utilities, but you should rely on the MRS User Manual to define the SSP utilities.

Section 6 - The Z80A Hardware

Overview

The Z80A card contains a complete computer system on a single card that plugs into the TI 99/4A Expansion Box chassis, including all the components, interconnections, and software needed to run CP/M-compatible application software. It includes the following components or subsystems:

- 4 Mhz Z80A CPU chip
- 64 Kb of dynamic RAM
- 8 Kb of ROM firmware expandable to 16Kb on-board
- two complete serial RS232 ports accessed via a standard DB25 connector
- two complete bidirectional parallel ports accessed via a 20 pin zero profile socket
- a floppy disk controller capable of controlling one to four single or double sided 40 or 80 tpi 5 1/4 inch floppy disk drives
- a standard Texas Instruments "clamshell" metal case designed specifically for the 99/4A expansion box
- all voltage regulators necessary to provide: +5 v for system power; +12v and -12 v for full-range RS232 voltage swings

CPU and Memory

The Foundation Z80A card is based on a 4Mhz Z80A 8-bit microprocessor chip. The CPU clock and all timing for the board are derived from a single one-phase 4 Mhz clock. The system is reset on power-up and when a momentary contact pushbutton switch is depressed. Other than taking power and ground connections from the TI expansion box, the Z80A card is totally independent of the 99/4A expansion bus. Note in particular that the Z80A system is not affected in any way by the reset signals on the expansion bus; the host 99/4A cannot reset the Z80A system. This design allows the Z80A system to continue running background processes while a host 99/4A system switches from one program to another.

The Z80A system uses standard 4164 dynamic RAM chips driven by a TMS4500A memory controller chip. Refreshes are performed synchronously during the Refresh interval of M1 CPU cycles, so they impose no overhead on the system. Also, since there is no DMA controller in the system, there is no possibility of extended bus use delaying a necessary refresh cycle. No wait states are inserted for DRAM access, permitting full 4 Mhz operation.

Bootstrap routines are contained in a single 2764 EPROM. Its socket is configured for compatibility with either 2764 or 27128 chips, allowing for possible future expansion to 16Kb of ROM-based software. At power up, the EPROM occupies the lowest 32Kb of CPU address space. A power-up/reset routine relocates an assembly-language monitor program and the MRS operating system BIOS into the upper 32Kb, populated by DRAM. When control is transferred to the monitor, the EPROM is shadowed out by setting a bit in the board command register. This bit can be reset at a later time by a program executing from the high 32Kb of memory, so that ROM-based software can be made available to disk-based programs. One wait state is inserted during EPROM accesses, which permits EPROMs with access times up to 450 nsec to be used.

A Z80A Counter/Timer Circuit (CTC) is used to provide four channels of counter or timer control. It is clocked at its maximum rate of 2 Mhz. The first two channels are used as baud rate generators for the serial ports, and the third is used as a watchdog timer for floppy disk operations by the operating system. Channel four can be used as a programmable timer by application programs.

Floppy Disk Controller

All floppy disk I/O is performed through a Western Digital 2793 floppy disk controller. This highly integrated chip contains all the features of its predecessor, the WD179x plus a high performance phase-lock-loop data separator and write precompensation logic. The WD2793 is capable of controlling either 5 1/4" or 8" disk drives. On the Foundation Z80A card, the 2793 is driven by a 2 Mhz clock which makes it capable of supporting 8" operation, but the ENMF Enable Mini-Floppy pin is hard wired to ground. Also, the operating system only provides disk drivers for 5 1/4" disk operation. The chip will support step rates of 6, 12, 20, and 30 msec. At present, the operating system preconfigures it for a 20 msec. step rate to maintain compatibility with Texas Instruments disk drives. The FDC will support single or double density, single or double sided 5 1/4 " drives. Operating system support is provided only for double density drives. In the SSDD mode, the floppy disk uses the Kaypro 2 disk format to gain access to software formatted for that popular machine. The remaining three formats retain most aspects of the Kaypro format but change sector interleaving to attain significantly faster access on multiple sector data transfers.

Serial I/O Ports

The Z80A card includes a Zilog DART or SIO/0 chip set up for asynchronous receive/transmit operations. Its basic function is as a parallel-to-serial and serial-to-parallel converter, but, within that role, its personality can be optimized for a given serial data communications application. The I/O pins of both DART channels come off the Z80A card through a single female DB25 connector wired to resemble a Texas Instruments RS232 'Y-connection'. The following table shows which of the DART signals are connected to the DB25 connector:

Signal Name	Input/ Output	DB25 Pin	Description
TxDA	Out	2	Transmitted Data, Channel A
RTSA*	Out	4	Request to Send, Channel A
RxDA	In	3	Received Data, Channel A
CTSA*	In	5	Clear to Send, Channel A
DTRA	Out	20	Data Terminal Ready. SEE BELOW.
		7	Signal Ground
TxDB	Out	14	Transmitted Data, Channel B
RTSB*	Out	18	Request to Send, Channel B
RxDB	In	16	Received Data, Channel B
CTSB*	In	13	Clear to Send, Channel B
DTRB	Out	19	Data Terminal Ready. SEE BELOW.

NOTE: The DTR signals feeding the DB25 connector are both permanently pulled high by a 2K resistor to +12v. The DTRA* and DTRB* pins on the DART are used in their capacity as general-purpose output bits to drive the HLT and ENP inputs to the 2793 FDC, respectively. RIA*, DCDA*, and DCDB* are all pulled down to ground. DCDB* is presently being used as an input bit to sense the disk Drive Ready* line. RxCA* and TxCA*, the clocks for channel A, are driven by the ZC/TO0 output of the CTC. RxTxCB* is driven by the ZC/TO1 output of the CTC.

The MRS operating system sets the Auto Enables bit in Write Register 3, which makes the DCD* and CTS* pins of the DART serve as receiver and transmitter enables. The DCD* pin is permanently asserted by being hardwired to ground; CTS* is driven through the DB25 connector, so the Z80A card will not transmit on channel A or B unless a terminal drives pins 5 and 13, respectively, of the DB25 connector to a positive voltage.

Interrupts

The powerful Z80A mode 2 interrupt is used by the Foundation Z80A card hardware and software. The interrupt hierarchy is as follows:

CTC interrupts -> PIO interrupts -> DART interrupts

with CTC interrupts having the highest priority. Standard Zilog IEI/IEO daisy chaining enforces this priority sequence. The FDC is interfaced solely by programmed I/O.

Parallel I/O Ports

The Z80A Parallel I/O device (PIO) provides two TTL-compatible programmable ports that can be used to interface peripheral devices to the Z80A card. Typical peripheral devices that can be interfaced through the parallel ports include keyboards, other CPU's, printers, SCSI-bus hard disk drives, and PROM programmers. No operating system support is provided for the parallel ports. The Z80A PIO includes the following major features:

- two independent 8 bit bidirectional peripheral interface ports with 'handshake' data transfer control
- interrupt driven handshake for fast response
- four modes of operation including byte output, byte input, byte-bidirectional (port A only), and bit control mode
- daisy chain priority interrupt logic
- TTL compatible inputs and outputs

On the Foundation Z80A card, the parallel ports are wired to a twenty pin socket that emerges near the etch fingers of the PC board on the non-component (solder) side of the board. This connector is labeled 'J3' on the silk screen legend. Pins are wired as shown in the following table (where pin numbers are those that would be assigned to an IC looking down on the component side of the board, i.e. pin 1 is nearest to the 12 v regulators and the etch fingers; pin 10 is nearest the 5 v regulator and the etch fingers; pin 11 is directly across from pin 10; and pin 20 is nearest to the array of 4164 memory chips):

Parallel Port Pinout Table

Pin Number	Signal Name	Purpose
1	A7	bit 7, channel A
2	A6	bit 6, "
3	A5	bit 5, "
4	A4	bit 4, "
5	A3	bit 3, "
6	A2	bit 2, "
7	A1	bit 1, "
8	A0	bit 0, "
9	ARDY	Channel A Ready handshake
10	ASTB*	Channel A Strobe (pulled up to +5 via 2Kohm)
11	B7	bit 7, channel B
12	B6	bit 6, "
13	B5	bit 5, "
14	B4	bit 4, "
15	B3	bit 3, "
16	B2	bit 2, "
17	B1	bit 1, "
18	B0	bit 0, "
19	BRDY	Channel B Ready handshake
20	BSTB*	Channel B Strobe (pulled up to +5 via 2 Kohm)

I/O Port Addresses

The following table lists the device addresses for the I/O ports on the Z80A system. Note that, as on most systems, I/O ports are not fully decoded.

Symbol	Address	Description
CREG	000H	Board Command Register (see below)
STAREG	010H	Disk Controller (FDC) Status Register
RDTRK	011H	FDC Read Track Register
RDSEC	012H	FDC Read Sector Register
RDDAT	013H	FDC Read Data Register
CMDREG	020H	FDC Command Register
WRTRK	021H	FDC Write Track Register
WRSEC	022H	FDC Write Sector Register
WRDAT	023H	FDC Write Data Register
ACTC	040H	Console port baud generator address
BCTC	050H	Auxiliary port baud generator address
CCTC	060H	CTC channel 3
DCTC	070H	CTC channel 4
CONDAT	080H	Serial port A data register address
CONCTL	084H	Serial port A control register address
AUXDAT	088H	Serial port B data register address
AUXCTL	08CH	Serial port B control register address
APDAT	0C0H	PIO Channel A Data
BPDAT	0C1H	PIO Channel B Data
APCTL	0C2H	PIO Channel A Control Register
BPCTL	0C3H	PIO Channel B Control Register

Board Command Register

An eight bit write-only register located at I/O address 0 latches the data bus to control the following functions. This register is cleared on power-up/reset.

Bit	Function
D7	EPROMEN* If clear, the EPROM occupies the low 32Kb of CPU address space.
D6	DDEN* If clear, the FDC is instructed to access the floppy disk as double density.
D5	SIDE1 If set, the floppy disk SIDE ONE* line is driven low, switching accesses from side 0 to side 1.
D4	DS1 If set, floppy disk 1 ("A:") is selected.
D3	DS2 Select floppy disk B:
D2	DS3 Select floppy disk C:
D1	DS4 Select floppy disk D:
D0	MOTOR If set, turn motor on in selected floppy disk drive.

MRS / OS

USER MANUAL

version 2.7

OCCO INCORPORATED
28 CLAFLIN STREET
MILFORD MA. 01757
(617) 478-3102

Copyright 1983,1985 OCCO inc.

TABLE OF CONTENTS

-GENERAL-

INTRODUCTION TO MRS....2
GETTING STARTED WITH MRS....2
USING THIS MANUAL....3
THE SOURCE DISK....4

-CONFIGURATION-

CONFIGURING MRS....5
MRS INITIALIZATION REQUIREMENTS....6
MODIFYING MRS I/O....7
MODIFYING THE MRS BOOTSTRAP....8
ASSEMBLING THE SOURCE CODE....9
LOADING AND STARTING MRS....9

-USER-

USING FILES UNDER MRS....10
MRS DISK DESIGNATION....11
MRS COMMANDS....12
MRS SUPERVISOR COMMANDS....13
SSP FUNCTIONS....14

-PROGRAMMING-

PROGRAMMING WITH MRS....36
MRS STRUCTURE....37
MRS MEMORY STRUCTURE....38
MRS CONSOLE I/O....39
MRS PRINTER I/O....40
MRS DISK I/O....41
MRS DISK ORGANIZATION....42
DIRECTORY STRUCTURE....43
FCB STRUCTURE....44
DIRECT DISK ACCESS....45
SEQUENTIAL FILE ACCESS....46
RANDOM FILE ACCESS....47
RUNNING USER PROGRAMS....48
CALLING MRS FROM A USER PROGRAM....49
MRS USER CALLS....50
PROGRAMMER'S QUICK REFERENCES....112

Copyright 1983 OCCO inc.

INTRODUCTION TO MRS

Welcome to the MRS system. MRS is system's software designed for the user. Minimal memory size, Z80 microprocessor efficiency and ease of use were all goals set before the design started. In addition, complete source listings to the operating system will be published to allow you to modify the system to your own needs. Please note that this document describes the MRS operating system as it is made available on many different computers. It is not tailored to describe specifically the implementation for the Foundation Z80A card. All sections of the manual continue to be true, however the version that is distributed with the Foundation Z80A card has already been adapted to run on that hardware; it need not be modified unless you so desire.

If you do want to modify the operating system yourself, please read the following sections carefully before attempting to install or modify MRS. We recommend that you configure and install the supervisor and utility package as they are delivered before attempting any customization to your particular tastes. Following this next section on configuring the operating system is a section entitled 'USER' that contains information of interest to anyone making use of MRS and a section entitled 'PROGRAMMING' that is appropriate to programmers who need direct access to the operating system calls.

GETTING STARTED WITH MRS

In order to modify, install, and use MRS in your computer system, you must have the following:

- Z80 microcomputer with at least 32K bytes of main memory addressed at 0000H to 7FFFH (hex).
- Terminal with I/O port
- At least one disk drive with controller
- Documentation for the terminal port & disk controller
- PROM or ROM bootstrap to load the MRS bootstrap from the disk.
- Editor and assembler for the Z80 microprocessor
- Some knowledge of Z80 assembly language

Copyright 1983 OCCO inc.

USING THIS MANUAL

This manual is divided into four basic parts. We recommend that you read ALL of the manual before attempting to configure or use MRS. On many pages, especially the technical sections, you will notice that a portion of the page is left blank. This was done in order to allow room for any notes or ideas of your own as well as to document any changes you may make to MRS.

A quick summary of the manual sections follows:

-GENERAL-	Introduction and general rules for using this manual and the source disk.
-CONFIGURATION-	The changes necessary to MRS to get it running on your computer.
-USER-	An explanation of MRS and SSP for the system user.
-PROGRAMMING-	How to use MRS in a programming environment.

NOTE: Throughout this manual hexadecimal numbers will be followed by the letter "H", all other numbers are decimal.

Copyright 1983 OCCO inc.

THE SOURCE DISK

Along with the manual you are supplied a 5.25" single sided double density floppy disk (KAYPRO 2 format).

There are four files provided that comprise the basic MRS distribution kit: MRS.SYS, MRS.BOT, SSP.COM, and FORMAT.COM. In addition, up to date notes are listed in file READ.ME, which can be accessed by typing "SSP TYPE READ.ME". This disk should be backed up immediately and used only if all other copies are destroyed.

Copyright 1983 OCCO inc.

CONFIGURING MRS

In order to modify the MRS system on your computer the following steps are necessary:

- Modify the MRSIO to correspond to the requirements of your computer's I/O
- Modify the MRS bootstrap
- Assemble MRS, SSP & MRS bootstrap
- Load & start MRS running
- Use SSP to initialize the system area of your disk
- Copy the system disk for backup purposes

Each step will be explained on the following pages.

Copyright 1983 OCCO inc.

INITIALIZATION REQUIREMENTS

Prior to starting MRS, in order for it to function correctly, your computer system must initialize the I/O that MRS will use. This step is left out of MRS because computers vary widely and most computer systems have PROM routines that initialize all I/O.

The following requirements are for a typical MRS system but are not necessarily identical to those for your system. The required functions may be swapped between the PROM routine and the DISK bootstrap buut must be present somewhere. In fact, afer you have MRS running, you may want to expand the DISK bootstrap to customize the computer to suit your tastes.

The source listing of the DISK bootstrap provided to you on the disk corresponds to the requirements below, that is, the bootstrap on the disk expects the PROM routine to perform the following functions:

-PROM BOOTSTRAP-

- -set terminal port baud rate
- -initialize terminal port
- -set printer port baud rate
- -initialize printer port
- -initialize disk controller
- -"home" drive A
- -load 1st sector, track 0 of disk A
- -jump to start of DISK bootstrap

-DISK BOOTSTRAP-

- -load rest of track 0 into memory
- -move head to track 1
- -load necessary sectors of track 1
- -clear register C
- -jump to start of MRS

Copyright 1983 OCCO inc.

MODIFYING MRSIO

In order to successfully mate the MRS system with your computer, the MRSIO system must "know" what your computer looks like. To do this it is necessary for you to modify two sections in the MRS source; the "EQUates" at the beginning of the listing and the MRS I/O routines themselves. In some cases, especially the console and printer I/O routines, only the EQUates may have to be modified as they provide the port addressing. In all cases read the discription of the specific routine you are modifing in the MRSIO source listing before you change anything. These sections tell you what the procedure must do to perform correctly.

The following is a descriptive flowchart of the procedure to modify an MRSIO routine. Follow this list for EACH routine.

- Read the requirements of the I/O procedure you are going to modify and make sure that you completely understand it.
- Read the documentation that came with your computer on the particular interface for which you are writing the procedure. List all of the addresses as you read.
- Write and assemble the procedure as a separate program.
- Test the procedure to make sure that it does what you intended and that the addresses are correct.
- Insert the procedure into the MRSIO in place of the existing one. Use the same label as the existing procedure or change the label in the jump table at the beginning of MRSSIO.

There will be some I/O which you will not have such as the reader or punch. These procedures may consist of a "RET" instruction only. In the future, when you are more familiar with the MRS system you may want to use these routines for other types of I/O.

NOTE: If you already have an I/O system compatable with CPM 2.2, then all that is necessary is that you provide the address of the BIOS to the MRS source in the EQUATES at the beginning of the listing. MRS is fully compatable with this type of BIOS.

Copyright 1983 OCCO inc.

MODIFYING THE MRS BOOTSTRAP

To modify the MRS bootstrap you should use the same flowchart as shown in the previous section on modifying MRSIO. The following are the requirements that this procedure needs in order to perform correctly:

- Load all of the sectors of the first disk track and the necessary sectors of the second track in order into the upper portion of main memory.

  NOTE: These sectors are not loaded by physical order but by logical order (see the sector table in the MRS BOOTSTRAP source listing).

- After completing the load from disk to memory the bootstrap procedure should jump to the start of MRS.

- The starting address of the bootstrap should be 80H (hex) unless your computer PROM bootstrap defines a different address into which the bootstrap is to be loaded.

- The entire bootstrap procedure, including any data tables, must be contained in the 1st sector of the disk.

- Any main memory which the bootstrap uses should not be in the same space that MRS occupies. Be especially careful about this in regards to the CPU stack pointer.

Copyright 1983 OCCO inc.

ASSEMBLING THE SOURCE CODE

There should be no great differences between the format of the source code supplied to you and what your assembler uses, however, there may be some slight changes required. The source code supplied uses the ZILOG mnemonics for the Z80 microprocessor.

When assembling the source code the "ORG" statement at the beginning of the code listing must be changed on the MRS and BOOT listings to conform to your system. Be conservative the first time time you assemble MRS and allow more than enough room. Later, after you see what the final memory address is for MRS, you may then adjust the "ORG" statement to use the minimum amount of memory.

The assembly address for SSP is supplied at 100H and should not be changed as this is where it will always be loaded and executed by the MRS SUPERVISOR.

NOTE: Some applications programs will require that the first byte used by MRS rests on a 256 byte boundary. It is recommended that you always assemble MRS on an address with the 8 lsb's = 0.

LOADING & STARTING MRS

After you have configured and assembled your MRS system, you will need to load it into memory and start it executing. There are two ways to do this. If it is possible (this is the preferred way) store the assembled MRS code and MRS bootstrap onto the disk starting at the 1st sector. The sectors should be loaded as they appear LOGICALLY on the disk not as they appear physically. Once this is done simply vector the CPU to the computer PROM bootstrap which will load the MRS bootstrap and start it executing.

If it is not possible or practical to load the assembled MRS code onto the system portion of the disk, then load it directly into the area of memory for which it was assembled. You may then start execution of the system by clearing register "C" and vectoring the CPU to the starting address of MRS. Once this is done you can use SSP to initialize the system portion of the disk (see SSP-INIT).

Copyright 1983 OCCO inc.

- File names consist of 1 to 8 alphanumeric characters followed by a period (".") and a 1 to 3 character extension. Upper and lower case characters are differentiated from each other by MRS. The extension is optional and, if not used, the "." character should be deleted.

  EXAMPLES:
  F
  FILENAME
  FILENAME.TX
  FILENAME.ASM

- Filenames are preceeded by an optional disk designator. The disk designator is a single character followed by a colon(":") which tells MRS the disk on which the file is located. If the disk designator is not used, the file is assumed to be on the current disk (see MRS DISK DESIGNATOR). Legal disk designators are A...P depending on the number of drives in your system.

  EXAMPLE:
  A:FILE2.ASM
  G:FILE2.ASM

- For some of the SSP commands, a universal file name may be used. This file name is constructed in the same fashion as explained above but, where, in a file name, any character will do, a "?" character may be used. In the case of several characters, the "*" may be used. WARNING: MRS will match any "?" or "*" with any character in a file name, BE CAREFUL that the files you specify with universal file names are the ONLY files that match the file name you have chosen.

  EXAMPLE:
  A:FILE?.A?M = A:FILE2.ASM,A:FILES.A6M
  B:FILE*.* = A:FILE12 ,A:FILE.ABC

- The following file names are reserved under MRS:

  - SSP.COM
  - BATCH.###

NOTE: If an illegal disk designator is used, MRS will abort the user program and display the appropriate message.

Copyright 1983 OCCO inc.

MRS DISK DESIGNATION

As stated previously, the disk designator is optional in a file name. In MRS there is a "current" disk. The current disk is where MRS will assume a file is located if no disk designator is specified with the file name. MRS always resets to this disk. The user can determine which disk is the current disk from the MRS prompt ("A." = disk A "B." = disk B etc.). The SSP prompt displays the current disk AFTER the period. The user may change the current disk by simply inputting ONLY the disk designator on a command line. This is especially helpful when working with SSP because of the housekeeping that must be done on a disk to remove or rename files. By switching the current disk to that particular disk, the user is less likely to accidentally erase the files of another disk. Also, the disk designator is not needed for files on that disk.

NOTE: Whenever you replace a disk in a drive or add one, be sure to use the CONTROL-C function to indicate to MRS that a new disk has been added to the system. If you do not, MRS will assume that the disk is the same as the former disk in the drive and may write over valid data on the disk.

Copyright 1983 OCCO inc.

There are two basic types of commands that MRS will receive and respond to; the supervisor codes which affect the immediate operating environment of the computer and the user program command lines which cause a user program to be loaded from a disk and executed.

The MRS supervisor codes, which are described in the next section, are control code inputs from the console which allow the user to control the console management features of MRS and perform other useful functions.

The MRS user command lines allow the user to call a program from the disk with a parameter line to direct the program's operations. These parameters include file names and switches. In general switches are characters preceeded by a slash character ("/") which tell the user program some particular option to be executed.

The general form of the user command line is:

PROGRAM_NAME/SWITCHES OTHER_FILENAME,OTHER_FILENAME

NOTE: The program name is a special case of the standard MRS file name. Regardless of whether or not an extension is used in the name, the extension "COM" will be inserted by MRS. Thus all executable files must use this extension. If a disk designator is used and the file is not found on that disk, MRS will search all lower disks for the file.

Copyright 1983 OCCO inc.

MRS SUPERVISOR COMMANDS

The user may communicate directly with the MRS SUPERVISOR through a set of control character commands. These commands are used to set certain parameters within MRS and otherwise to control the system operation. MRS interprets these commands directly and does NOT pass them onto the user program under normal circumstances. If however, the user program is using direct console I/O either by directly accessing the console port or using the direct I/O request (call 6), then the MRS SUPERVISOR will NOT receive or implement the commands.

- ^C - When the MRS system detects this character from the console, it will immediatly vector CPU control to the appropriate routine (see call CNTRC).

- ^P - This command causes the MRS SUPERVISOR to echo all output to the console on the printer. Since the output is echoed after any console I/O is performed, the complete console I/O functions (see CONSOLE I/O) are reflected at the printer.

- ^Q - This command turns off the printer echo.

- ^R - At times it may be desired to vector the CPU to ROM control for use as a monitor/debugger or to reboot the system. As soon as this character is received by MRS, a jump to a user defined location is executed.

- ^S - This command causes the MRS SUPERVISOR to halt output to the console every 24 lines. The system will then wait for any character (not a command charcter) from the console before continuing the output. This character will NOT be reflected to the user software.
  NOTE: After this command is received, the following carrage return will cause MRS to hold for a character.

- ^T - This command disables the ^S function discribed above.

- ^U - In some systems the output from an assembler or linker is left in user memory and is required to be saved on disk. This command which is executed ONLY if the MRS internal monitor is running (no user program running) will cause all of the user memory to be moved up 8K bytes. This allows SSP to be loaded and to SAVE any necessary user data (see SSP-SAVE).

- ^V - During BATCH operation it may be desired to terminate the mode early but, not in the middle of the currently executing program. This command forces the BATCH CODE byte to zero which will terminate the BATCH at the end of the current program (see SSP-BATCH).

- ^W - Restart the user program now in memory.

Copyright 1983 OCCO inc.

SSP is a collection of utility programs which were designed to provide all the necessary support to the user to maintain his/her computer. Instead of providing separate utilities which take up more disk and memory space, all the utilities are grouped together and are coded independently enough to allow the programmer to add his/her own support programs easily.

The specific utilities provided in SSP are described in the following pages. You should read these carefully before using them to prevent any misunderstanding between what you expect the utility to do and what it actually does.

SSP is called into memory in the same fashion as any user program:

SSP <return>

or

SSP command <return>

In the first case SSP will, once loaded and executing, respond with the SSP prompt "SSP.X" (where X = the current disk drive character). The user may then type on the console the appropriate SSP command. If SSP responds with an error message the user may type "HELP" to have SSP display the command menu. SSP may ONLY be aborted with the command "END" as it sets the CONTROL-C function to its own internal routine. This is done so that the user may stop the current command from executing by using CONTROL-C without aborting SSP.

The second method of using SSP is when the user desires to have only one command executed from SSP and then return control to the MRS supervisor. This second method is also used by the MRS BATCH operator (see SSP-BATCH).

NOTE: SSP was designed to operate under the MRS/OS system ONLY. It will not successfully operate for most of the commands under any other operating system. While it does not "know" anything about the internal organization of MRS/OS it does use many of the calls which only MRS provides.

Copyright 1983 OCCO inc.

BATCH
BATCH <file>

The MRS system can be run under the user interactive mode (the normal operating mode) or the BATCH mode. In the BATCH mode, a predetermined set of command lines, stored in a disk file are automatically loaded and executed by MRS using the BATCH operator in SSP. These command lines may be stored in a permanent or temporary file on the disk depending on the way that the BATCH operator was called.

- mode 1: BATCH INTERACTIVE INPUT -

In this mode the BATCH operator is called without a reference to any file name in the command line. The BATCH operator will open a file on the current disk (which can be any disk) called "BATCH.###". The BATCH prompt will appear and the user should type in the command lines he/she wishes in the order that they are to be executed. To signal to the BATCH operator that all of the lines have been input, the operator should enter a <return> only on the last line. If SSP is in interactive mode the BATCH operator will return with a message indicating that BATCH operations have been enabled and then the SSP prompt will appear again. After the SSP command "END" is entered by the user, MRS will begin BATCH operations.

- mode 2: BATCH COMMAND FILE INPUT -

The BATCH operator may also execute a set of commands from a prestored disk file. This is often used when a set of MRS commands are frequently used and the user does not desire to wait to type each one in individually. This effectively does the same thing as the BATCH INTERACTIVE mode but instead of taking its input from the console the BATCH operator loads its operating file from the prespecified disk file and returns to either MRS or SSP immediately.

NOTE: When using the BATCH mode to execute SSP commands you should use the SSP name in front of the command as you would if you were typing the command in yourself.

EXAMPLE: Batch>SSP PRINT FILENAME

NOTE: If a fatal error is encountered during BATCH operations, the BATCH mode will be aborted.

NOTE: The user may abort the batch mode by typing the MRS command "^V" during BATCH operations. This will allow the current command to complete operation but stop subsequent BATCH operations.

Copyright 1983 OCCO inc.

BOOT

This command forces control of the CPU to go to the cold boot address listed in the jump table at the front of the MRSIO source code. Normally this command will never be used, but if the user desires to reboot MRS after BATCH operations or similar circumstances, this command may be used.

Copyright 1983 OCCO inc.

CMPR FILENAME1,FILENAME2

This command is used to compared two files together. It is also used to get an exact record count of a file by comparing the file against itself. Each file is read in, 8K bytes at a time and compared against the corresponding bytes in the other file. If the files are of different sizes, the SSP will respond with the appropriate message. If the files are of the same size the resultant message will list the total number of errors encountered. The order that the filenames in the command line are entered does not matter, but there must be two file names.

NOTE: If SSP encounters any errors during the execution of this command, the BATCH mode will NOT be affected - BATCH operations WILL continue if they are operating.

NOTE: If the file length is over 65,535 records, the total number of records listed in the CMPR message will be the total amount minus 65,535 (eg. a file with 65,545 records in it will show in the message as having 10 records).

Copyright 1983 OCCO inc.

COPY disk1,disk2

This command is used when it iis desired to copy the contents of an entire disk onto another (usually for backup purposes). The copy process begins on track 0 and continues through the last track, one track at a time. Data is copied from disk 2 to disk 1. The copy command does not verify the disk contents after writing to the disk.

NOTE: Since the COPY command totally destroys any information that was previously on disk1, it provides you with an opportunity to verify that you really want to copy the disk before starting execution. It does this by asking if you are ready to perform the copy operation. A negative response will abort the COPY command.

NOTE: At least one disk must be specified in the command line. If only one disk is specified, the COPY procedure will ask you for the source and destination disks. Up to 250 records are read/written at a time to minimize sswap requirements.

NOTE: Copy requires that both the source and the destination disk be of the same size and format.

Copyright 1983 OCCO inc.

DIR
DIR disk
DIR filename

This command allows the user to see the entire contents of a disk directory or any part of it he/she wishes. The directory is loaded into memory and a search is made for the desired files. The sorted files are then displayed on the console along with other statistical information.

- mode 1: COMPLETE DISK DIRECTORY -

If no disk designator or file name follows the DIR command, the entire directory of the current disk is displayed and examined.

- mode 2: COMPLETE DISK DIRECTORY NON-CURRENT DISK -

If a disk designator follows the DIR command, the entire directory for that disk will be displayed and examined.

- mode 3: FILE SPECIFIC DIRECTORY -

If a file name follows the DIR command, ONLY those directory entries which correspond to the specified file name will be displayed and examined.

NOTE: In mode 3 type commands, the file name may have universal characters in it. The file name "*.*" is the same as using the directory command without the file name.

NOTE: If the mode 3 command is used, the dissk statistics apply ONLY to the particular files specified.

NOTE: If the disk directory is longer than the console screen can handle, use the MRS supervisor code "^S" to display part of the directory at a time.

NOTE: See "PDIR" command for special print mode of the DIR command.

Copyright 1983 OCCO inc.

SSP - DUMP (file in hex)

DUMP filename,start address (hex)

This command displays the specified disk file in hexadecimal format. The display is broken up into lines of 16 bytes each with the address of the first byte on the line diisplayed in hex followed by each byte separated by a space and displayed in the hex format. The last 16 characters of the line contain the ASCII equivalent of the byte if it is a printable character. Non-printable characters are shown with a "." character.

To allow the user to better interpret the file dump when it is binary code, the DUMP command requires a hexadecimal starting address. This is the address that the first byte in the file will be assigned, all other bytes follow correspondingly.

NOTE: See the "PDUMP" command for a printing form of this command.

SSP - END

END

This is the command to terminate SSP and return to MRS when SSP is operated in the interactive mode.

NOTE: This is the only command which will return the CPU control to the MRS supervisor. SSP uses the CONTROL-C function to allow the user to terminate an SSP command without leaving SSP control.

Copyright 1983 OCCO inc.

SSP - PROTECT

PROTECT disk
PROTECT filename

This command allows the user to write protect either a disk or a file. The disk is protected until the next MRS reset occurs. A file is protected by setting a bit in the disk directory and thus, is write protected until released (see "RELEASE").

- mode 1: PROTECT DISK

MRS will set a bit in memory corresponding to the specified disk. If, after that bit is set, a write is attempted to the disk, MRS will abort the user program and display the appropriate message. This write protect feature is also available to user programs.

- mode 2: PROTECT FILE

Individual files may be write protected by specifying a filename instead of a disk. The major difference between this mode and mode 1 is that the file is protected by setting a bit on the disk which is not reset except by the release command. MRS will abort any user program which attempts to write to the file. (note: a write protected file may be renamed and transfered but cannot be erased).

SSP - RELEASE

RELEASE filename

This command is used to remove the write protection from a file.

Copyright 1983 OCCO inc.

ERA filename

This command allows the user to eliminate any files from the disk he/she chooses. All files corresponding to the specified file name are marked as unused on the disk directory. The command terminates with a message indicating the number of disk directory FILES that were erased.

NOTE: Universal file names can be used here but BE CAREFUL. If the file name "*.*" is used the entire directory will be erased.

NOTE: If files on the disk have accidentally been erased and the DISK HAS NOT BEEN WRITTEN TO the following procedure can be used to restore those files (read the section SSP-PATCH before proceeding):

-1- Call the PATCH command with the appropriate disk as the PATCH parameter.

-2- Read in each directory block and, if a file to be restored is located in that block, do steps 3 & 4.

-3- Clear the directory status byte using the PATCH "M" command for all entries corresponding to the file to be restored.

-4- Write the record back to the SAME sector on the disk.

-5- When you have finished use the DIR command to make sure that no clusters have been allocated twice.

-6- Use the CONTROL-C function to reallocate the disk.

Copyright 1983 OCCO inc.

This command displays the SSP command menu as shown below.

SYSTEM SUPPORT PROCEDURES (SSP) MENU

COMMAND	PARAMETERS	USE
END	none	terminate SSP
BOOT	none	vector to machine PROM
TYPE	file	display file
PRINT	file	print file
DUMP	file,hxaddr	display file in hex
PDUMP	file,hxaddr	print file in hex
REN	file1,file2	rename file2 using file1 name
ERA	file	erase file from directory
PATCH	disk/file	edit disk or file in hex
SAVE	file,rsize	save user memory
LOAD	file	load file into memory
INIT	disk,file	init disk OS space
DIR	<disk/file>	display the disk directory
PDIR	<disk/file>	print the disk directory
XFER	file1,file2	transfer file2 to file1
CMPR	file1,file2	compare file2 with file1
COPY	disk1,disk2	transfer disk2 to disk1
MERGE	file1,file2	merge file2 into file1 (ASCII)
BATCH	<file>	execute batch mode
PROTECT	disk/file	write protect the disk/file
RELEASE	file	remove the file write protection
CHKDSK	file	check disk for errors

Copyright 1983 OCCO inc.

INIT disk,filename

This command initializes the system portion of the disk (see MRS DISK ORGANIZATION) specified in the command line. SSP expects to find 2 files existing on the specified disk with the filename. The first file should have the same name with the file extension of ".BOT". This is the MRS bootstrap file in object code format. The SSP will load the first file and write it to the first sectors of track 0 of the specified destination disk. The second file should have the same file name with the extension of ".SYS". ALL of this file will be loaded into memory and written to the rest of the system portion of the destination disk. At the termination of the command, a message will be written with the total number of sectors in the system area of the destination disk that were actually written.

NOTE: The file name on the parameter line of this command does not require an extension. SSP will insert the necessary extensions automatically. Be careful that the two files used have the same name and the required extensions.

Copyright 1983 OCCO inc.

MERGE file1,file2

This utility merges two ASCII files together. SSP will read in and scan the first file until it encounters an EOF (1A hex) character. It will then start copying the second file into the first until a second EOF character is found. A sector of EOFs is written to the end of the first file before it is closed.

Copyright 1983 OCCO inc.

PATCH disk
PATCH filename

This command allows the user to directly view and alter the contents of the disk. The particular portions of the disk which are to be used are specified in the command line. The actual coommands for the PATCH utility are quite simple and aree explained in the next section.

- mode 1: PATCH DISK -

By specifying only a disk, the entire disk (all sectors) can be patched.

- mode 2: PATCH filename -

If the user wants to limit the patch session to a single file he/she can specify only the file name.

Copyright 1983 OCCO inc.

SSP - PATCH - COMMANDS

The PATCH utility contains five commands for file manipulation, a menu command, and an exit command. The use of all these commands is explained below.

COMMAND: R rec#

The Read command reads a logical record from the disk to the PATCH buffer. The record number entered is relative to the start of the entity being patched (i.e.: disk, or file). At the time PATCH is invoked it will display the valid range of record numbers to be used for either the Read or Write commands.

NOTE: When scanning disk sectors it is not necessary to continually use the read command. Initialize the PATCH pointer by using the read command on the first sector you want to read and then simply press <return> to read the next sector in sequence. PATCH will automatically update the pointer.

COMMAND: W rec#

The Write command is the only PATCH command that actually modifies the contents of the disk. It will take the contents of the PATCH buffer and copy it to the specified record. Write does not check to make sure the buffer is being written to the same record from which it was loaded so- BE CAREFUL!!

NOTE: If a record read from the disk has not been modified it is not necessary to write it back to the disk.

COMMAND: M offset,h#,..

The Modify command is used to alter the contents of the PATCH buffer. You must specify the offset from the start of the buffer to the first byte you wish to modify. Follow this with a string of hexadecimal byte values seperated with commas for all sequential bytes to be changed. PATCH will make the changes and automatically display on the console a verification of the changes made.

NOTE: The Modify command only changes the contents of the buffer and has no affect on the data on the disk. In order to actually modify the data on the disk you must use the Write command to write the modified buffer back to disk.

Copyright 1983 OCCO inc.

COMMAND: A offset,string

The Ascii command provides a convient method for modifying the contents of the buffer with ASCII data. It functions in an analogous fashion to the Modify command except instead of using hex data, an ASCII character string is inserted directly. This feature eliminates the need to manually do an ASCII to HEX conversion.

COMMAND: I hex#

The Initialize command allows you to initialize an entire block to a single selected value. The Initialize command modifies only the PATCH buffer and has no affect on the contents of the disk.

COMMAND: H

The Help command types on the console screen the PATCH help menu as shown below:

```
PATCH commands
Rrec#            read record (rec#$)
Wrec#            write record (rec#)
Moffset,H#,..    modify offset bytes to hex values
Aoffset,string   place ASCII string at offset
Ihex#            initialize block to hex#
H                display help message
E                end PATCH & return to SSP exec
```

COMMAND: E

End PATCH utility and return to SSP executive.

Copyright 1983 OCCO inc.

SSP - PDIR (print directory)

```
PDIR
PDIR disk
PDIR filename
```

This command is basically the same as the DIR command but causes the disk directory to be printed instead of displayed on the console. It is included to allow a hard copy of the directory to be stored with the disk.

Copyright 1983 OCCO inc.

PDUMP filename,start address

This is the same basic command as DUMP execept that the output goes to the printer instead of the console(see SSP-DUMP).

Copyright 1983 OCCO inc.

PRINT filename

SSP will output the ASCII file to the printer. Because the PRINT rouutine uses the TYPE routine and simply steers the output to the printer, the supervisor commands will work during the execution of this command (CONTROL-S, CONTROL-T etc.).

NOTE: At the end of the print command two form feeds will be sent to the printer and the printer output will be turned off regardless of it's previous state.

Copyright 1983 OCCO inc.

REN file1,file2

SSP will search for the file corresponding to file2 and, if found, will rename it with the file name used in the file1 parameter of the command line.

NOTE: You may use universal characters in both file names but be careful! The SSP will find all file names which match the 2nd file name and transfer all file name characters to the first file name when not explicitly stated in the command line.

Copyright 1983 OCCO inc.

SAVE filename,size

The contents of user memory starting 8K bytes above the start of user memory will be saved on disk using the file name specified. The number of bytes saved corresponds to the amount specified in the command line. This value is specified in DECIMAL in RECORDS (1 record = 128 bytes).

NOTE: This command is used in conjunction with the MRS supervisor code " ^U " which shifts the user memory up 8K bytes. This function has been provided because there are some application software packages which leave data in memory that must be saved on disk. The supervisor command has been provided to make enough room in memory for SSP to be loaded without destroying the user data to be saved.

NOTE: When using this command, it should be the FIRST command given to SSP once it is loaded into memory. There are a number of other commands in SSP which use the data area and will destroy the user data.

Copyright 1983 OCCO inc.

SSP - LOAD

LOAD file

The contents of user memory is LOADed with the specified file. After the file is LOADed, the SSP will display the actual number of records loaded.

This command is used for two purposes, to transfer a file from one disk to another using one drive and to limit the size of a file. Both uses involve the LOAD and SAVE commands.

NOTE: To transfer a file from one disk to another use the following sequence of operations:

- LOAD the file from the source disk.
- SWAP the disks.
- Enter CONTROL-C to allocate the new disk.
- SAVE the file on the destination disk using the record count given by the LOAD procedure.

NOTE: To limit the size of a file, LOAD the file and SAVE it specifying a smaller number of records.

NOTE: The actual size of the file LOADed is dependent on the amount of memory available. In a 64K system, approx. 340 records will be LOADed.

Copyright 1983 OCCO inc.

TYPE file

This command outputs the contents of the specified file to the console. It is assumed that the file is in ASCII format.

NOTE: It was specifically for this command that the MRS supervisor code "^S" was implemented. By using the screen halt function the user can examine a file without printing it.

Copyright 1983 OCCO inc.

XFER file1,file2

This command transfers file2 to file1. The first file name is created on the disk and the second file is read 8K bytes at a time, and written to the first file. After the file transfer has completed, the first file is closed, a message is printed indicating that the transfer is complete and the files are then compared by automatically implementing the CMPR command. If the file names are identical and on the same disk the command will fail.

NOTE: If the two file names are identical but on DIFFERENT disks, the following command line format may be used:

XFER d1:=d2:filename

where d1 & d2 are the respective disk designators.

Copyright 1983 OCCO inc.

SSP - CHKDSK

CHKDSK file

This command will check a disk and allocate any clusters found in error to the file name specified in the command line. The file name MUST include a disk designator. At the end of the check procedure, the SSP will display the number of clusters found in error. The clusters are checked by reading each record. Normally this command is used only on newly formatted disks prior to storing any data on them.

DO NOT USE THIS COMMAND ON ANY DISK WITH A DIRECTORY IN USE.

The 1st and succeeding records (if necessary) of the disk directory are destroyed by this procedure. If your BIOS asks for directions when an error is encountered, simply tell it to continue.

NOTE: As every record is read on the disk, this procedure may take several minutes to run on some disks. This is especially true if a large number of errors are encountered.

Copyright 1983 OCCO inc.

PROGRAMMING WITH MRS

The following sections will guide you in using MRS for programming in assembly language. It is assumed that you have some experience using Z80 assembly language and have MRS running on your computer. Where it is felt helpful the user is referred to the SSP source code for working examples of MRS programming. The user should look through the source code in any case, to get a good idea of how to use MRS and its multiple user requests.

MRS is designed to provide the interface between user programs and your computer. There are three primary interfaces that MRS will manage; the console I/O, the printer I/O and the disk I/O. The various sections of this manual which follow will describe in detail how you, as a writer of user programs, can get the most out of your computer using MRS.

Copyright 1983 OCCO inc.

MRS STRUCTURE

The MRS system is divided into four primary components:

- MRS SUPERVISOR
- MRS I/O
- MRS BOOTSTRAP
- MRS UTILITIES (SSP)

The SUPERVISOR contains the brains of the system. It is a collection of software procedures which provide the overall control of the computer. What program, what input/output and what is legal/illegal in the operation of the computer are all determined here. The SUPERVISOR is normally resident whenever the MRS system is running.

The I/O component provides the interface between the other components of the MRS system and yoour computer. It is a collection of procedures that handle requests to transmit or receive data from the peripherals and mass storage devices connected to the computer. The I/O section of MRS is normally resident in memory whenever the MRS system is running.

The BOOTSTRAP is used to start the MRS system. It normally resides in the system portion of the disk and is loaded into main memory only when needed. The BOOTSTRAP loads the configured MRS system and starts it executing.

The final component of the MRS system is the utilities package or SSP (System Support Programs). This software also normally resides on the disk and is called into memory only when needed. It provides the user with the necessary functions to manage the computer and it's mass storage.

Copyright 1983 OCCO inc.

MRS MEMORY STRUCTURE

The general memory structure for the MRS system is shown below. Absolute addresses are given where possible. Because the specific addresses of where the MRS system will reside are different for each system it is not possible to illustrate all of the addresses in absolute form. The USER area of memory is not used by the MRS system and will not be affected by MRS except the HIGHEST 128 BYTES OF USER MEMORY WILL BE USED WHENEVER A USER PROGRAM IS NOT RUNNING. During user program execution, the system stack space (which is internal to MRS) is used and NO USER STACK SPACE IS REQUIRED.

address	use
0-100H	system command & interface area (see below)
100H-USERTOP	user memory for programs & data
USERTOP+1-MEMORYEND	MRS resident area

low memory address	use
0	jump to system warm boot
5	user-to-MRS call address
5CH	1st user FCB address
6CH	2nd user FCB address
80H	system command line and default DMA address

Copyright 1983 OCCO inc.

MRS CONSOLE I/O

OUTPUT

Output by a user program to the console can be handled in three ways; direct output (often used to set console display characteristics), single character output (used for displaying the character "$" and when nonlinear data is displayed) and buffered output (terminated by a "$").

Direct output bypasses the MRS console supervisor and sendss the data directly to the console as 8 bit bytes. No attempt is made by MRS to count lines, use the CONTROL-S screen halt function or steer the I/O to the printer.

Single character output performs the same as buffered output but does not include the decimal and hexadecimal character translation features of buffered output.

Buffered output supports all the MRS console related commands (screen halt, printer echo, CONTROL-C etc.) as well as providing special number translation (see call 9). Output to the console using buffered I/O is terminated with the character "$".

INPUT

Input from the console to the user program can also be handled in three ways; direct input (used to bypass MRS commands), single character input (for special editing) and buffered input (using MRS editing capabilities).

Direct input bypasses all MRS command functions and returns with the 8 bit character from the console. MRS makes no attempt to interpret or use the character in any way.

Single character input is checked by MRS for command codes and may, or may not be echoed as the user wishes. All control codes that are not MRS commands are returned and, when echoed, are written to the console with a tilda ("~") preceeding the character, which is trranslated into upper case prior to display.

Buffered input provides editing of the line as well as character echo. Character deletion with stop at the end of the buffer (no deletion over end of buffer) is also handled.

Copyright 1983 OCCO inc.

MRS PRINTER I/O

There are two methods of using the printer I/O under MRS. Each method will be described below but it is recommended that you read the section "CONSOLE I/O" before proceeding.

STANDARD ACCESS

This method of printer output uses the standard printer call to MRS. MRS will make no attempt to manage the output but will check the console input for MRS command characters (CONTROL-C etc.). All output to the printer will be as 8 bit bytes as given to MRS by the user program. The version of the MRS I/O that is in the source code contains the code necessary for a Centronix parallel interface used by many printers.

STEERED ACCESS

The second method of implementing print output under MRS is to use the console manager in MRS and steer the output to the printer. This is done by making a call to MRS prior to output which directs MRS to send the output to the printer and/or the console (see call SETIO). Once this call is implemented the user may then output data to the printer using the console output calls (see CONSOLE I/O).

Copyright 1983 OCCO inc.

MRS provides the user with three ways to use the disk storage available to the computer; direct disk access, sequential disk file access, and random disk file access. Each method of access is described fully in the following sections.

Basically MRS allows the user to view the disk storage in one of three ways:

-DIRECT ACCESS- The disk is made up of logical sectors (or records) of 128 bytes each. The user specifies to MRS which disk, which sector, and the direction of transfer (read or write).

-SEQUENTIAL- On the disk is an array of data which the user may gain access to in a sequential fashion, 128 bytes at a time. This array is referred to as a FILE of data. The user specifies to MRS the file type before starting the access and the location of the File Control Block (FCB) during the access. MRS will take care of the details from there.

-RANDOM- On the disk is an array of data as in the sequential access method above. The user must specify to MRS the same parameters as above but may now access the 128 byte blocks in any order that he/she chooses.

 Copyright 1983 OCCO inc.

MRS DISK ORGANIZATION

MRS divides the disk into three sections discribed below:

-SYSTEM AREA- This section occupies the first tracks of the disk and is used to hold the MRS bootstrap and MRS/OS. It is present on all disks regardless of whether they are to be used as system disks or not. The SYSTEM AREA is initialized with the SSP command "INIT" (see SSP-INIT).

-DIRECTORY- MRS uses this section of the disk to keep traack of user files and disk allocation. The directory resides on the first logical sectors of the third track of the disk. It is present on all disks and should not under ANY circumstances be altered by the user until he/she has had some experience with MRS.

-USER AREA- This section is the major portion of the disk and is used to hold the user files. MRS uses the DIRECTORY to keep track of the location and size of each user file.

Copyright 1983 OCCO inc.

DIRECTORY STRUCTURE (single density disk)

The directory occupies sixteen 128 byte sectors or 2K bytes on the disk. There are 64 directory entries available in this space. Each entry is responsible for keeping track of up to 16K bytes of the disk space. If a file requires more than 16K bytes than a second entry can be allocated to the file. Every entry in the disk directory which is used by a file is referred to as an extent of that file. Thus a file of 3K bytes will have one extent, or entry, in the directory while a file with 38K bytes will have three extents.

The following is a description of a directory entry:

BYTE	USE
0	entry status: 0 = used E5 = unused (open)
1-8	file name
9-11	file extension (see noote below)
12	extent number
13-14	unused (reserved for MRS)
15	file size: number of 128 byte records in entry
16-31	file allocation in clusters

The disk is divided into clusters for use by MRS in disk space allocation. Each cluster consists of eight sectors in a logical sequence on the disk. The first 2 tracks on the disk are not considered, thus there are a total of 243 clusters available. The first two clusters are allocated to the directory, this leaves clusters 2 to 241 for user files.

NOTE: Byte 9 uses the most significant bit (7) to store the write protection status of the file. If bit 7 is set, the file may only be read from. Any attempt to write or erase a write protected file will cause MRS to abort the user program.

Copyright 1983 OCCO inc.

FCB STRUCTURE

In order to use MRS for managing a user's files during access, the user must retain a File Control Block (FCB) in the user portion of memory. This block must be 33 bytes long for sequential access of the file or 36 bytes long for random access of the file. The FCB consists of the directory entry (see description below) and the pointer to the file record being accessed.

In the case of the sequential access pointer, a single byte is used which points at the cluster being accessed and the individual record within that cluster.

In the case of random access, the last 3 bytes in the FCB are used to point to a specific record within the file and the 33rd byte is not used.

FCB MAKEUP

BYTE	USE
0	disk designator (0=current,1="A",2="B" etc.)
1-8	file name
9-11	file extension
12	extent number
13-14	reserved for MRS
15	number of records in entry
16-31	cluster array
32	sequential record pointer
33-35	random record pointer

Copyright 1983 OCCO inc.

DIRECT DISK ACCESS

MRS allows the user to access a disk using direct access to the disk sectors. This method of reading and writing to disk is not recommended for use by the beginner and only for maintenance purposes by anyone. The SSP-PATCH and SSP-COPY utilities should be used whenever possible and also serve as a good example of how to use direct disk access.

Basically, the disk is divided into sectors or records for this type of access. The user specifies the disk drive, the record number and the type of access (read or write). MRS will check the sector number to be sure it is a valid number, translates it into the track and logical sector numbers, translates the logical sector number into the physical sector number and executes the requested operation on the disk. NO CHECK is made by MRS on the area of the disk being accessed by the user program so - !!!BE CAREFUL!!!.

Copyright 1983 OCCO inc.

SEQUENTIAL FILE ACCESS

This is the normal type of access most user programs will make when accessing disk data. File management for read and write operations is provided by MRS and the user does not have to get involved. A good example of this type of disk access is seen in the SSP-XFER source code. The following steps illustrate the necessary steps for a user program to access a disk file:

-1- Format the file name to the user FCB using the FORMAT call. This places the file name in the FCB in a pattern that MRS can easily recognize.

-2- Create the file on the disk using the CREAT call

or (if the file already exists)

Open the file using the OPEN call.

In either instance MRS will setup your FCB for access by loading in the disk directory entry and clearing the sequential file access byte (1st cluster & 1st record).

-3- Read or write to the file keeping in mind that the sequential pointer is moved forward AFTER each file access regardless of whether the access was a read or a write, that is, if you do 2 read accesses of the file and then a write, you will have written to the third record of the file (pointer = 2). Also you should know that MRS checks the status of the pointer BEFORE it executes the requested operation, so if the end of the file extent (FCB) is reached MRS will open a new extentt WHEN THE ACTUAL READ OR WRITE TO THAT EXTENT IS REQUESTED. This saves directory space by not opening an extent that will not be used.

-4- Close the file by using the CLOSE call. This copies the FCB to the disk directory. This step is not necessary if the file was only read.

NOTE: While a file is being accessed, the only way that MRS "knows" about the file is through the FCB. DO NOT ALTER THE FCB!!!.

Copyright 1983 OCCO inc.

RANDOM FILE ACCESS

This is the most common method of access for database and record keeping programs. It enables the user to go directly to the desired record while skipping all records in between. This access method uses the random record pointer bytes of the FCB (see FCB STRUCTURE). The file can be thought of as a large array of data blocked into 128 byte boundaries. The user simply stipulates which block is to be accessed and MRS provides the necessary file and disk management to execute the request. There is no limit on the particular record number specified other than it must be within the size of the disk.
If a record is specified to be written and there exists empty records or extents that normally preceed thhat record, the extent and cluster for that record will be opened regardless. This implies that some files will have a "virtual" size. That is, they may have 1st, 2nd, and 4th extents but no 3rd extent.

The necessary steps to access a file are the same as for sequential access (see SEQUENTIAL FILE ACCESS) except that the random record pointer in the FCB is used (see FCB STRUCTURE).

NOTE: During random file access, MRS will not increment the random record pointer. This step is left to the user. MRS WILL update the sequential record pointer to point to the record just accessed as well as updating the FCB as necessary. This allows the user to specify a starting record number for sequential file access.

NOTE: After a random disk access the sequential record pointer points to THE RECORD JUST READ and not to the next record. Therefore, if a sequential read follows a random read the same record is read twice. In order to avoid this condition the user program should increment the sequential record pointer if a sequential access is to follow a random access.

Copyright 1983 OCCO inc.

RUNNING USER PROGRAMS

All user application programs are loaded and started by MRS starting at location 100H (hex). The following system parameters are set by the MRS SUPERVISOR prior to transfering CPU control to the user software:

- The control-C (^C) address is set for the MRS internal function which will abort the user proogram and reset the system if received from the console.

- The disk buffer address is set at 80H (hex). If the disk I/O is used at this address, the command line will be overwritten on a disk read.

- The CPU Stack Pointer (SP) is set to 100H (hex). It is recommended that the user execute the following instruction as one of the 1st opcodes in his program:

  ```
  LD      SP,(6)
  ```

  This will set the SP to the highest possible location (+1) that is available to the user. Since the PUSH opcode causes an automatic predecrement of the SP prior to storing data on the stack, this instruction alone is sufficient to initialize the SP.

- The command line which caused the user program to be loaded and executed is stored in memory starting at location 80H. The first location contains the number of characters which were in the command line MINUS THE USER PROGRAM NAME. The name of the user program is also NOT within the command line:

  ```
  console cmd line - FILE1 ABCD/G
  command line     - 7, ,A,B,C,D,/,G,0
  ```

  In this example "FILE1" is loaded into memory and executed by the MRS SUPERVISOR and the rest of the command line is left for the user.

- The first file name encountered (if any) after the command file name is formated and located at location 5CH (hex). This location may be used for an FCB if so desired.

Copyright 1983 OCCO inc.

CALLING MRS FROM A USER PROGRAM

There is a single standard method for calling the MRS request codes from a user program. This technique allows the user to make use of the services provided by MRS without having to define external references or linking to a special function library.

Basically, all requests are called by placing the specific request call number in register "C" and then performing a CALL to address 0005H. Some requests also require parameters to be passed or returned in one or more of the other registers in the CPU. These are all specified below in the separate reference sections for each request.

Copyright 1983 OCCO inc.

REQUEST: RESET (RESET)

INPUT PARAMETERS: C=0

OUTPUT PARAMETERS: none

FUNCTION: Reset MRS and return control to the operating system.

COMMENTS: This call and the jump to location 0 are interchangeable. Both return control to the MRS monitor. If the BATCH mod is enabled, the SSP will be loaded into user memory and control given to it following this call.

EXAMPLE: Return control to MRS.

```
LD      C,0
CALL    5

   or

JP      0
```


Copyright 1983 OCCO inc.

REQUEST: RDCE (READ CHARACTER WITH ECHO)

INPUT PARAMETERS: C=1

OUTPUT PARAMETERS: A=character

FUNCTION: Read a character from the console, write the character to the console display and return with the character in register A.

COMMENTS: MRS will wait for a character from the console and, upon reading it, reset the msb and write it back to the console. MRS command characters will NOT be returned or echoed. All other control characters will be returned and echoed as explained previously.

EXAMPLE: Read the console and wait for a character "R".

```
RDLOOP:     LD      C,1
            CALL    5
            CP      A,'R'
            JR      NZ,RDLOOP
```


Copyright 1983 OCCO inc.

REQUEST: WRTC (WRITE CHARACTER)

INPUT PARAMETERS: C=2
E=character

OUTPUT PARAMETERS: none

FUNCTION: Write the character in register E to the console.

COMMENTS: Control characters will be written as explained under CONSOLE I/O.

EXAMPLE: Write an "R" to the console.

```
LD      C,2
LD      E,'R'
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: READR (READ READER)

INPUT PARAMETERS: C=3

OUTPUT PARAMETERS: A=character

FUNCTION: Read byte from reader.

COMMENTS: Not implemented in MR5IO as shipped. On return from this call the value in register A is undefined.

Copyright 1983 OCCO inc.

REQUEST: PUNCH (PUNCH)

INPUT PARAMETERS: C=4
E=character

OUTPUT PARAMETERS: none

FUNCTION: Write byte to punch.

COMMENTS: This function is not implemented in MRSIO as shipped.
No register contents are changed by this call.

Copyright 1983 OCCO inc.

REQUEST: PRTC (PRINT CHARACTER)

INPUT PARAMETERS: C=5
E=character

OUTPUT PARAMETERS: none

FUNCTION: Write the character in register E to the printer.

COMMENTS: The character as represented in register E, is written to the printer with no check for control characters.

EXAMPLE: Write the character "R" to the printer.

```
LD      C,5
LD      E,'R'
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: DIRIO (DIRECT I/O)

INPUT PARAMETERS: C=6
E=ASCII character (for output)
or
E=-1 (for input)

OUTPUT PARAMETERS: none (for output)
or
A=character (for input)

FUNCTION: Provide direct input/output to the console. If the value in register E is NOT a -1 then that value is sent directly to the console (as 8 bits). If the value in register E is a -1 then the next character received from the console is returned in register A.

COMMENTS: When this function is used for input, the MSB of the character from the console is cleared.

EXAMPLE: Send an "R" to the console and wait for a "^A".

```
        LD      C,6
        LD      E,'R'
        CALL    5
LOOP:   LD      E,-1
        CALL    5
        CP      A,01H
        JR      NZ,LOOP
```


Copyright 1983 OCCO inc.

REQUEST: RDIO (READ I/O)

INPUT PARAMETERS: C=7

OUTPUT PARAMETERS: A=I/O byte

FUNCTION: Return with the I/O byte in register A.

COMMENTS: This call is for future versions of MRS that may implement multitasking. For the time being, the byte at location 3 in memory serves as the I/O byte.

EXAMPLE: Read the I/O byte.

```
LD      C,7
CALL    5
LD      (IOBYTE),A
```


Copyright 1983 OCCO inc.

REQUEST: WRIO (WRITE I/O)

INPUT PARAMETERS: C=8
E=I/O byte

OUTPUT PARAMETERS: none

FUNCTION: Set the I/O byte.

COMMENTS: This call is for future use in MRS.
For now, the byte at memory location 3 serves as the I/O byte.

EXAMPLE: Set the MRS I/O byte.

```
LD      C,8
LD      A,(IOBYTE)
LD      E,A
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: WRBUF (WRITE BUFFER)

INPUT PARAMETERS: C=9
DE=address of 1st character in buffer

OUTPUT PARAMETERS: none

FUNCTION: Write the character string pointed to by register DE to the console. The last character in the string is a "$" which is NOT written to the console.

COMMENTS: All characters are written to the console with control characters written as previously explained. Two special switches are available for user use. These are unprintable characters which, when encountered in the character stream, affect the handling of the next 1 or 2 characters.

CHARACTER	USE
128	The next character is written in hex form.
130	The next 2 characters are assumed to be an unsigned decimal number. This number is written with leading zeros suppressed. If the number is less than 10,000 then it is written left justified. If the number is zero then one zero is written.

EXAMPLE: Write the character string to the console.

```
        LD      C,9
        LD      DE,CHRSTR
        CALL    5
        .
        .
        .
CHRSTR: DB      128,21H,' is written as ',130,21H,0,'$'
```

The output will look like:

"21 is written as 33"

Copyright 1983 OCCO inc.

REQUEST: RDBUF

(READ TO BUFFER)

INPUT PARAMETERSS: C=10
DE=address of 1st character in buffer

OUTPUT PARAMETERS: 2nd byte in buffer=actual buffer size

FUNCTION: Input from the console a character string into the buffer. The input function is terminated by a carriage return or when the total input count exceeds the count in the 1st byte of the string. The actual character count is written into the 2nd byte in the string (the third byte has the 1st character input from the console). If the input is terminated by a carriage return, the carriage return is NOT written to the buffer. In its place an ASCII null character (byte=0) is written.

COMMENTS: All characters are echoed to the console. Control characters are echoed by preceeding the character with a "^" and printing the control character with the upper ASCII bit set to produce an upper case equivalent character.

EXAMPLE: Read the following input from the console.

```
        LD      C,10
        LD      DE,INBUF
        CALL    5
        .
        .
        .
INBUF:  DB      0AH,FFH,FFH,FFH,FFH,FFH,FFH,FFH,FFH,FFH
```

input character string = '12AB'(control-D)(return)

echoed string on console = '12AB^D'

buffer after input = 0AH,05H,31H,32H,41H,42H,04H,0,FFH,...

Copyright 1983 OCCO inc.

REQUEST: CSTA (CONSOLE STATUS)

INPUT PARAMETERS: C=11

OUTPUT PARAMETERS: A=0 (receiver empty)
or
A=-1 (receiver full)

FUUNCTION: Test status of console input receiver and return appropriate parameter.

COMMENTS: The status of the receiver is tested for input. NO check is made to see if the input is a valid ASCII character or if it is a control character. If the character is an MRS control character and function RDCE is requested immediately after this call, MRS will execute the control character function and then wait for another character from the console.

EXAMPLE: Test for an input from the console and, if there, get it.

```
LD      C,11
CALL    5
OR      A,A
JR      Z,AHEAD         ;CONTINUE IF NO INPUT
LD      C,2             ;GET INPUT
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: VERSN (VERSION NUMBER)

INPUT PARAMETERS: C=12

OUTPUT PARAMETERS: HL=MRS version number

FUNCTION: Return the current MRS version number to the user.

COMMENTS: This call is used to allow the user program to determine if the current MRS level will support the system calls that may be used.

EXAMPLE: Check that the MRS version will support the multiply and divide functions.

```
LD      C,12            ;GET VERSION #
CALL    5
LD      A,17H           ;1.7 & BELOW=NO MULT/DIVD
CP      A,L             ;EARLIER VERSION?
JR      NC,NOMULT       ;YES-NO MULTIPLICATION
.
.
.
```


Copyright 1983 OCCO inc.

REQUEST: DSKRST (DISK RESET)

INPUT PARAMETERS: C=13

OUTPUT PARAMETERS: none

FUNCTION: Mark ALL disks as unallocated.

COMMENTS: The write protect bit for each drive is reset also.

EXAMPLE: Reset the entire disk system and end program.

```
LD      C,13
CALL    5
JP      0
```

NOTE: It is not necessary to use this call before ending a user program. MRS will implement this function automatically on a warm boot.

Copyright 1983 OCCO inc.

REQUEST: DSKSEL (DISK SELECT)

INPUT PARAMETERS: C=14
E=drive number

OUTPUT PARAMETERS: none

FUNCTION: Select a drive as the current drive.

COMMENTS: All ambiguious drive references are then made to the selected drive. If the drive selected is NOT within the current MRS configuration, the user program will be aborted and an appropriate message written to the console. The following are the legal drive numbers for an MRS system at maximum configuration:

drive A = 0
.
.
P = 15

EXAMPLE: Select drive C.

```
LD      C,14
LD      E,2
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: OPEN (OPEN FILE)

INPUT PARAMETERS: C=15
DE=address of FCB

OUTPUT PARAMETERS: A=directory block# (file found)
or
A=-1 (file not found)
DE=address of directory entry in memory

FUNCTION: Find the file specified in the FCB. If the file exists on the disk, the directory entry is copied into the FFCB.

COMMENTS: See the appropriate section for references to the FCB. This call may be used to determine if a file exists before using the CREATE function.

EXAMPLE: Open the file specified by the FCB. If the file is not in the disk directory, use the CREATE function to open a new directory entry.

```
        XOR     A               ;INIT THE FCB
        LD      (FCB+12),A      ;EXTENT#
        LD      (FCB+32),A      ;& RECORD#
        LD      C,15            ;OPEN FILE
        LD      DE,FCB
        CALL    5
        CP      A,-1            ;FILE FOUND?
        JR      NZ,AHEAD        ;YES-CONTINUE
        LD      C,22            ;NO-ENTER INTO DIRECTORY
        LD      DE,FCB
        CALL    5
        .
        .
        .
FCB:    DB      0,'FILNAME TYP',0,0,0,0
        DS      17
```


Copyright 1983 OCCO inc.

REQUEST: CLOSE

(CLOSE FILE)

INPUT PARAMETERS: C=16
DE=address of FCB

OUTPUT PARAMETERS: A=directory block# (if entry found)
or
A=-1 (if entry not found)

FUNCTION: Locate the file entry, specified by the FCB, and copy the FCB onto the disk there. The extent numbers must match.

COMMENTS: This function is automaticly performed when, during a sequential WRITE operation, the extent is filled. The user may terminate file I/O by using this call but it is not necessary unless the file has been written to.

EXAMPLE: Close the file specified by the FCB after data has been written to the disk.

```
        LD      C,16            ;CLOSE THE FILE
        LD      DE,FCB
        CALL    5
        .
        .
        .
FCB:    DB      0,'FILNAME TYP',0,0,0,0
        DS      17
```


Copyright 1983 OCCO inc.

REQUEST: SRCH (SEARCH FOR FIRST ENTRY)

INPUT PARAMETERS: C=17
DE=address of FCB

OUTPUT PARAMETERS: A=directory block# (if file found)
or
A=-1 (if not found)
HL=entry address (if file found)

FUNCTION: Initialize the file search parameters and look for the disk entry corresponding to the FCB.

COMMENTS: When searching for a disk entry, this call MUST be executed before using the SRCHN function. This call will find the first entry encountered in the directory and SRCHN finds subsequent entries.

EXAMPLE: Find the entry corresponding to the FCB and test for it's directory status.

```
        LD      C,17            ;FIND ENTRY
        LD      DE,FCB
        CALL    5
        CP      A,-1            ;ENTRY FOUND?
        JR      Z,NOENT         ;NO-JUMP
        .
        .
        .
FCB:    DB      0,'FILNAME TYP',0,0,0,0
        DS      17
```


Copyright 1983 OCCO inc.

REQUEST: SRCHN (SEARCH FOR NEXT ENTRY)

INPUT PARAMETERS: C=18
DE=address of FCB

OUTPUT PARAMETERS: A=directory block# (if entry found)
or
A=-1 (if not found)
HL=address of entry in memory

FUNCTION: Locate the next directory entry corresponding to the FCB. Start the search immediately after the last entry searched.

COMMENTS: This call is used AFTER the function SRCH to locate additional files in a disk directory. NO OTHER CALLS USING THE DISK SHOULD BE USED BETWEEN SUBSEQUENT SRCH AND SRCHN CALLS. MRS may alter the search parameters if other calls are used. This call is designed to be used with the UPDAT function to alter the disk directory as necessary.

EXAMPLE: Locate the next file associated with an FCB.

```
        LD      C,18            ;FIND ENTRY
        LD      DE,FCB
        CALL    5
        CP      A,-1            ;ENTRY FOUND?
        JR      Z,NOENT         ;NO-JUMP
        .
        .
        .
FCB:    DB      0,'FILNAME TYP',0,0,0,0
        DS      17
```


Copyright 1983 OCCO inc.

REQUEST: DELET (DELETE ENTRY)

INPUT PARAMETERS: C=19
DE=address of FCB

OUTPUT PARAMETERS: A=number of directory entries deleted

FUNCTION: Locate all directory entries which match the FCB and mark them as unused entries in the directory.

COMMENTS: This call may be used to erase multiple files in the directory by using the universal character ("?"). The total number of ENTRIES (not files) erased will be returned in register A.

EXAMPLE: Delete all files with the name "FILE1" regardless of the extension.

```
        LD      C,19            ;DELETE FILES
        LD      DE,FCB
        CALL    5
        LD      (MESG+1),A      ;GIVE # TO MESSAGE
        LD      C,9             ;WRITE MESSAGE TO CONSOLE
        LD      DE,MESG
        CALL    5

FCB:    DB      0,'FILE1   ???',0,0,0,0
        DS      17

MESG:   DB      130,0,0,' ENTRIES WERE DELETED$'
```


Copyright 1983 OCCO inc.

REQUEST: RDREC (READ RECORD)

INPUT PARAMETERS: C=20
DE=address of FCB

OUTPUT PARAMETERS: A=0 (if read is successful)
or
A<>0 (if end of file encountered)

FUNCTION: Read the next sequential record as indicated by the sequential record pointer of the FCB. The disk data will be read into the disk buffer space (see call BUFSET).

COMMENTS: If an error is encountered reading the disk, control will NOT return to the user. An error message will be sent to the console and control transfered to the MRS monitor.

EXAMPLE: Read the next record and return to MRS if the file is empty.

```
        LD      C,20            ;READ RECORD
        LD      DE,FCB
        CALL    5
        OR      A,A             ;READ SUCCESSFUL?
        JP      NZ,0            ;NO-END PROGRAM
        .                       ;YES-CONTINUE
        .
        .

FCB:    DB      0,'FILE1    .TXT',0,0,0,0
        DS      17
```


Copyright 1983 OCCO inc.

REQUEST: WRREC (WRITE RECORD)

INPUT PARAMETERS: C=21
DE=address of FCB

OUTPUT PARAMETERS: A=0 (if write successful)
or
A<>0 (no more space)

FUNCTION: Write the record indicated by the sequential record pointer of the FCB to the disk. The data will be taken from the disk buffer area (see call BUFSET). of the FCB to the disk.

COMMENTS: If an error is encountered in writing the data to the disk, an error message will be sent to the console and control will be transferred to the MRS monitor.

EXAMPLE: A subroutine to write data to the disk and return if the transfer was successful. If the transfer was not successful, return to MRS.

```
        PUSH    DE              ;SAVE REGISTERS
        PUSH    BC
        LD      C,21            ;WRITE DATA
        LD      DE,FCB
        CALL    5
        OR      A,A             ;WRITE SUCCESSFUL?
        JP      NZ,0            ;NO-RETURN TO MRS
        POP     BC              ;YES-RESTORE REGISTERS
        POP     DE
        RET                     ;DONE

FCB:    DB      0,'FILE1   TXT',0,0,0,0
        DS      17
```


Copyright 1983 OCCO inc.

REQUEST: CREAT (CREATE FILE)

INPUT PARAMETERS: C=22
DE=address of FCB

OUTPUT PARAMETERS: A=directory block # (if file created)
or
A=-1 (if directory full)

FUNCTION: Create a file in the disk directory using the data in the user FCB. The directory entry and FCB are initialized for the 1st extent. The FCB is also initialized with the disk # and the sequential record pointer.

COMMENTS: If a file already exists using the name in the FCB, the file and all it's entries are deleted first. If the user wants to check whether or not a file exists, the user should first attempt to use the OPEN call.

EXAMPLE: If the current file does not exist on the disk, create it.

```
        LD      C,15            ;ATTEMPT TO OPEN FILE
        LD      DE,FCB
        CALL    5
        CP      A,-1            ;FILE OPENED?
        RET     NZ              ;YES-RETURN
        LD      C,22            ;NO-CREATE IT
        CALL    5
        CP      A,-1            ;FILE CREATED?
        JP      Z,0             ;NO-RETURN TO MRS
        .
        .
        .

FCB:    DB      0,'FILE1   TXT',0,0,0,0
        DS      17
```


Copyright 1983 OCCO inc.

REQUEST: RENAM (RENAME FILE)

INPUT PARAMETERS: C=23
DE=address of FCB

OUTPUT PARAMETERS: A=number of entries renamed

FUNCTION: Find all entries matching the 1st file name in the FCB and replace the names in the directory with the 2nd file name in the FCB.

COMMENTS: Only one file at a time should be renamed (no general character "?" should be used). The second file name in the FCB starts at "FCB+16" and uses the same file name format as the first name. The disk specifier in the 2nd name is ignored.

EXAMPLE: The following subroutine will rename the file specified in FCB #1 with the name in FCB #2.

```
        PUSH    HL              ;SAVE REGISTERS
        PUSH    DE
        PUSH    BC
        LD      HL,FCB2         ;XFER 2ND NAME
        LD      DE,FCB1+16      ;TO 1ST FCB
        LD      BC,12
        LDIR
        LD      C,23            ;RENAME FILE
        LD      DE,FCB1
        CALL    5
        POP     BC              ;RESTORE REGISTERS
        POP     DE
        POP     HL
        RET                     ;DONE,A=# ENTRIES RENAMED

FCB1:   DB      0,'FILE1   TXT',0,0,0,0
        DB      0,'NEWNAME EXT',0,0,0,0,0

FCB2:   DB      0,'FILE2   TXT',0,0,0,0
        DS      17
```


Copyright 1983 OCCO inc.

REQUEST: LOGIN (LOGGED IN DISKS)

INPUT PARAMETERS: C=24

OUTPUT PARAMETERS: HL=bit map of disks currently logged in.

bit 0 = disk A

.

.

bit 15= disk P

FUNCTION: Provide the user with a bit map of all disks currently logged in by MRS.

COMMENTS: If a disk is logged in (MRS has a clusterr bit map for it) the corresponding bit in HL will be set.

EXAMPLE: Check to see if disk B is logged in.

```
DSKA:   EQU     01H             ;DISK A LOGIN MASK
DSKB:   EQU     02H             ;DISK B
DSKC:   EQU     04H             ;DISK C
DSKD:   EQU     08H             ;DISK D

        LD      C,24            ;GET LOGIN PARAMETER
        CALL    5
        LD      A,L
        AND     A,DSKB          ;DISK B LOGGED?
        JR      NZ,BON          ;YES
        JR      BOFF            ;NO
```


Copyright 1983 OCCO inc.

REQUEST: CURDSK (CURRENT DISK)

INPUT PARAMETERS: C=25

OUTPUT PARAMETERS: A=current disk #

A = 0 = disk A

.

.

A = 15= disk P

FUNCTION: Return to the user the number of the currently selected default disk.

COMMENTS: If the value of the disk designator in an FCB is 0, the default disk is used for the requested function. This call tells the user which disk that is.

EXAMPLE: Write the value of the current disk to the console.

```
        LD      C,25            ;GET CURRENT DISK VALUE
        CALL    5
        ADD     A,41H           ;CONVERT TO ASCII CHARACTER
        LD      (MESG),A        ;GIVE TO MESSAGE
        LD      C,9             ;WRITE MESSAGE
        LD      DE,MESG
        CALL    5
        .
        .
        .
MESG:   DB      0,' IS THE CURRENT DISK$'
```


Copyright 1983 OCCO inc.

REQUEST: BUFSET (BUFFER SET)

INPUT PARAMETERS: C=26
DE=disk buffer address

OUTPUT PARAMETERS: none

FUNCTION: Change the disk I/O record buffer address to the new value in DE.

COMMENTS: During disk read/write operations the memory buffer address (where the data is written to/read from) is set by this call. When a user program is started by MRS, the buffer address is set to 0080H. This address need not be changed by the user but keep in mind that the buffer will overrun the command line normally used by MRS. If disk I/O is performed prior to using the command line parameters from the user, the parameters will be lost unless the buffer address is changed by using this call.

EXAMPLE: Change the disk buffer area to the user buffer.

```
        LD      C,26            ;CHANGE BUFFER ADDRESS
        LD      DE,DSKBUF
        CALL    5
        .
        .
        .
DSKBUF: DS      128             ;128 BYTE BUFFER SIZE
```


Copyright 1983 OCCO inc.

REQUEST: ALOVCT (ALLOCATION VECTOR)

INPUT PARAMETERS: C=27

OUTPUT PARAMETERS: HL=address of cluster allocation map

FUNCTION: Return information regarding the current disk.

COMMENTS: This call is used by some utility routines and should NOT be needed by the user.

Copyright 1983 OCCO inc.

REQUEST: WRPRT (WRITE PROTECT)

INPUT PARAMETERS: C=28

OUTPUT PARAMETERS: none

FUNCTION: Mark the current disk as write protected until the next warm boot.

COMMENTS: Once this call is made any attempt to write to the disk will abort the user program in the same fashion as a disk error. Only on a warm boot, at the end of the user program, will the write protect be removed.

EXAMPLE: Write protect the current disk.

```
LD      C,28
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: ROVCT (READ ONLY VECTOR)

INPUT PARAMETERS: C=29

OUTPUT PARAMETERS: HL=read only bit map

FUNCTION: Return the bit map of the disks currently write protected.

COMMENTS: This function is used to allow the user program to determine which drives have read only access rights. If the bit corresponding to the drive iis set, then that drive is write protected.

```
BIT 0 = drive A
        .
        .
BIT 15= drive P
```

EXAMPLE: Check to see whether drive C is write protected.

```
LD      C,29
CALL    5
BIT     2,L
JP      NZ,WRTPRT       ;JUMP IF PROTECTED
```


Copyright 1983 OCCO inc.

REQUEST: SETFLG

INPUT PARAMETERS: C=30
DE=FCB address

OUTPUT PARAMETERS: A=entry count

FUNCTION: Set the write protect bit in the directory for the file(s) listed in the FCB.

COMMENTS: Used to protect files from being modified.

EXAMPLE: Write protect all "FILE1.*" files.

```
        LD      C,29
        LD      DE,FCB
        CALL    5

FCB:    DB      0,'FILE1   ???'
        DS      24
```


Copyright 1983 OCCO inc.

REQUEST: GETDPB

INPUT PARAMETERS: C=31

OUTPUT PARAMETERS: HL=DPB Address

FUNCTION: Get the address of the current disk's DPB.

COMMENTS: Used to obtain data about the current disk.

EXAMPLE: Get and store the current disk's DPB address.

```
        LD      C,31
        CALL    5
        LD      (DPBADR),HL
DPBADR: DW      0               ;ADDRESS OF DPB
```


Copyright 1983 OCCO inc.

REQUEST: USRCD

(USER CODE)

INPUT PARAMETERS: C=32
E=user code (set user code)
or
E=-1 (read user code)

OUTPUT PARAMETERS: none (set user code)
or
A=user code (read user code)

FUNCTION: Read or set the user code. If register E is a -1 then return with the current user code in register A. If register E is not -1 then set the user code with the value in register E.

COMMENTS: This call is commonly used to allow the transfer of a parameter from one program to another. The initialized value of the user code on MRS startup is 0.

EXAMPLE: Read the user code and increment it.

```
LD      C,32
LD      E,-1
CALL    5
INC     A                ;INCR THE CODE
LD      E,A              ;AND SET THE NEW VALUE
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: RDFIL (READ FILE)

INPUT PARAMETERS: C=33
DE=FCB address

OUTPUT PARAMETERS: A=0 (read successful)
or
A<>0 (end of file encountered)

FUNCTION: Read the file specified in the FCB using the random access record pointer. The disk data will be read into the user disk buffer (see call BUFSET).

COMMENTS: If an error is encountered during the read operation, the user program will be aborted. If the read is successful, the FCB and the sequential record pointer will be updated as needed.

EXAMPLE: Read record 100 of a file and reeturn to MRS if not read.

```
LD      C,33
LD      DE,FCB
LD      HL,100
LD      (FCB+33),HL
CALL    5
OR      A,A
JP      NZ,0
```


Copyright 1983 OCCO inc.

REQUEST: WRFIL (WRITE FILE)

INPUT PARAMETERS: C=34
DE=FCB address

OUTPUT PARAMETERS: A=0 (write sucessful)
or
A<>0 (no space left)

FUNCTION: Write a record to the disk specified in the FCB using the random record pointer in the FCB.

COMMENTS: If an error in encountered in writing to the disk, the user program will be aborted. If the write is successful, the FCB and sequential record pointer will be updated as necessary.

EXAMPLE: Write record 100 and, if out of disk space, return to MRS.

```
LD      HL,100
LD      (FCB+33),HL
LD      C,34
LD      DE,FCB
CALL    5
OR      A,A
JP      NZ,0
```


Copyright 1983 OCCO inc.

REQUEST: COMSIZ (COMPUTE SIZE)

INPUT PARAMETERS: C=35
DE=FCB address

OUTPUT PARAMETERS: random record pointer=last record# +1

FUNCTION: Compute the total size of the file specified in the FCB and store that value into the random record pointer in the FCB.

COMMENTS: The actual value returned by this call is the last record number in the file +1. This is the total "virtual" file size (the actual record size may be considerable smaller due to unused records). The FCB is NOT updated by this call nor is the sequential record pointer.

EXAMPLE: Find the file size and write to the console.

```
        LD      C,35
        LD      DE,FCB
        CALL    5
        LD      HL,(FCB+33)     ;GET FILE SIZE
        LD      (MMSG+1),HL
        LD      C,9             ;WRITE MESSAGE
        LD      DE,MSG
        CALL    5

MSG:    DB      130,0,0,' BYTES IN FILE$'
```


Copyright 1983 OCCO inc.

REQUEST: COMREC

(COMPUTE RECORD NUMBER)

INPUT PARAMETERS: C=36
DE=FCB address

OUTPUT PARAMETERS: random record pointer=random record#
(sequential record#)

FUNCTION: Compute the random record number that corresponds to the sequential record number.

COMMENTS: The random record number is computed from the FCB's sequential record pointer and extent#. This call is often used to keep track of certain records during a sequential scan of a file.

EXAMPLE: Get the random record number and save it in an array.

```
LD      C,36
LD      DE,FCB
CALL    5
LD      IY,(ARRAY)        ;ARRAY POINTER
LD      HL,(FCB+33)       ;NEW RECORD NUMBER
LD      (IY+0),L
LD      (IY+1),H
INC     IY
INC     IY
LD      (ARRAY),IY
```


Copyright 1983 OCCO inc.

REQUEST: DSKRST (DISK RESET)

INPUT PARAMETERS: C=37

OUTPUT PARAMETERS: none

FUNCTION: Deallocate all disks and remove the write protection.

COMMENTS: This call may be used after alterations have been made to disk directories. The function is automatically implemented on a warm boot.

EXAMPLE: Reset the disk system.

```
LD      C,37
CALL    5
```


Copyright 19833 OCCO inc.

REQUEST: NOT IMPLEMENTED

INPUT PARAMETERS: C=38

Copyright 1983 OCCO inc.

REQUEST: NOT IMPLEMENTED

INPUT PARAMETERS: C=39

Copyright 1983 OCCO inc.

REQUEST: WRZRO (WRITE ZEROES)

INPUT PARAMETERS: C=40
DE=FCB address

OUTPUT PARAMETERS: A=0 (write successful)
or
A<>0 (no space left)

FUNCTION: Write a record of zeros to the record pointed to in the FCB random record pointer.

COMMENTS: The user buffer will be cleared by this call and the rest of the call will be treated as a random record write (see WRFIL).

EXAMPLE: Write a cleared record to the file.

```
LD      C,40
LD      DE,FCB
CALL    5
OR      A,A             ;WRITE SUCCESSFUL?
JP      NZ,0            ;RETURN TO MRS IF NOT
```


Copyright 1983 OCCO inc.

REQUEST: RDC (READ CHARACTER NO ECHO)

INPUT PARAMETERS: C=128

OUTPUT PARAMETERS: A=character from console

FUNCTION: Wait for and read a character from the console but do not echo it.

COMMENTS: This call would typically be used to input a password from a user or to provide output other than the input character to the console.

EXAMPLE: Get 2 character password from user. Abort program if wrong character is received.

```
        LD      C,9             ;WRITE MESSAGE OUT
        LD      DE,PSWMSG
        CALL    5
        LD      C,128           ;GET 1ST CHARACTER
        CALL    5
        CP      A,PSWC1         ;1ST CHARACTER OK?
        JP      NZ,0            ;NO-ABORT PROGRAM
        CALL    5               ;GET 2ND CHARACTER
        CP      A,PSWC2         ;2ND CHARACTER OK?
        JP      NZ,0            ;NO-ABORT PROGRAM
        .
        .
        .

PSWMSG: DB      'ENTER TWO CHARACTER PASSWORD $'
PSWC1:  EQU     'A'
PSWC2:  EQU     'Z'
```


Copyright 1983 OCCO inc.

REQUEST: NOT IMPLEMENTED

INPUT PARAMETERS: C=129

Copyright 1983 OCCO inc.

REQUEST: CNTRC (CONTROL-C)

INPUT PARAMETERS: C=130
DE=user control-C routine address
or
DE=-1 (disable control-C function)
or
DE=0 (abort user program on control-C)

OUTPUT PARAMETERS: none

FUNCTION: Set the routine address for control-C or disable the function.

COMMENTS: This call is used when the user wishes to provide his own routine for the control-C function. CPU control is passed to the user with all user registers intact and the previous, or last, user PC value on the stack. The control-C input may be ignored entirely using address (-1) or may be set to the MRS control-C routine using address (0). The MRS routine (address=0) is always selected when the user program is first started.

EXAMPLE: Set the control-C address for the user routine.

```
        LD      C,130           ;SET 'C ADDRESS
        LD      DE,CPROG
        CALL    5
        .
        .
        .

;USER CONTROL-C ROUTINE

CPROG:  POP     HL              ;GET LASST PC VALUE
        .
        .
        .
```


Copyright 1983 OCCO inc.

REQUEST: RDLOG

(READ LOGICAL RECORD)

INPUT PARAMETERS: C=131
B=disk# (0=current,1=A...15=P)
DE=logical record #

OUTPUT PARAMETERS: A=0 (read successful)
or
A=3 (illegal record #)
or
A=-1 (read failed)

FUNCTION: Read the logical record specified in DE from the disk specified in B. The logical record number is assumed by MRS to be linear that is, the record number is independent of the physical location on the disk.

COMMENTS: This is a first method of access to the disks using MRS. Normally, only utility routines would use this call but the user may wish to for disk access under special circumstances (eg. encripted storage).
A return parameter value of 3 indicates that the record number is out of range for the disk. Single density, 8" disks have a range of 0 to 2001 logical records. Note that this allows access to several records on the disk which the user would ordinarily NOT have access to. These include the system area of the disk (logical record numbers 0 to 51) and left over records (logical record numbers 1972 to 2001).

EXAMPLE: Read a logical record from the current disk.

```
        LD      C,131           ;READ RECORD
        LD      B,0
        LD      DE,(RCDNUM)
        CALL    5
        OR      A,A             ;RECORD READ?
        JR      NZ,HGHNUM       ;NO-NUMBER TOO HIGH
        .
        .
        .
RCDNUM: DW      0               ;LOGICAL RECORD NUMBER
```


Copyright 1983 OCCO inc.

REQUEST: WRLOG (WRITE LOGICAL RECORD)

INPUT PARAMETERS: C=132
B=disk# (0=current,1=A...15=P)
DE=logical record #

OUTPUT PARAMETERS: A==0 (write successful)
or
A=3 (illegal record #)
or
A=-1 (write failed)

FUNCTION: Write the logical record specified in DE to the disk specified in B. The logical record number is assumed by MRS to be linear that is, the record number is independent of the physical location on the disk.

COMMENTS: This is a first method of access to the disks using MRS. Normally, only utility routines would use this call but the user may wish to for disk access under special circumstances (eg. encripted storage).
A return parameter value of 3 indicates that the record number is out of range for the disk. Single density, 8" disks have a range of 0 to 2001 logical records. Note that this allows access to several records on the disk which
this allows access to several records on the disk which
the user would ordinarily NOT have access to. These include the system area of the disk (logical record numbers 0 to 51) and left over records (logical record numbers 1972 to 2001).

EXAMPLE: Write a logical record to the current disk.

```
        LD      C,132           ;WRITE RECORD
        LD      B,0
        LD      DE,(RCDNUM)
        CALL    5
        OR      A,A             ;RECORD WRITTEN?
        JR      NZ,HGHNUM       ;NO-NUMBER TOO HIGH
        .
        .
        .
RCDNUM: DW      0               ;LOGICAL RECORD NUMBER
```

 Copyright 1983 OCCO inc.

REQUEST:CLSTIO (CLUSTER I/O ENABLE)

INPUT PARAMETERS: C=133
E=0 (disable cluster I/O)
or
E=1 (enable cluster I/O)

OUTPUT PARAMETERS: none

FUNCTION: Disable/enable actual data reads and writes to all disks.

COMMENTS: This request is used when directtory updates are required but actual data I/O is not. MRS will return a zero in register "A" to indicate that the operation was successful even though it was not performed. A disk reset call (call 13) will enable cluster I/O as will a reset at the end of the program. Logical disk operations (calls 131 and 132) are NOT affected by this call. See the SSP - CHKDSK code for an example of this call.

EXAMPLE: Disable cluster I/O.

```
LD      C,133          ;SETUP MRS CODE
LD      E,0            ;FLAG=DISABLE I/O
CALL    5              ;DISABLE CLUSTER I/O
.
.
.
```


Copyright 1983 OCCO inc.

REQUEST: FORMAT (FORMAT FILE NAME)

INPUT PARAMETERS: C=134
DE=address of FCB
HL=address of file name

OUTPUT PARAMETERS: none

FUNCTION: Format a file name into an FCB. The file name may or may not have a disk specifier and an extension.

COMMENTS: This call is used to set up an FCB after obtaining a file name (typically from the console). If a disk specifier is used the proper disk number will be setup, otherwise the default disk designator (0) will be used. The universal character ("*") will be expanded to fiill in the remaining characters in either the file name or extension. Note that the byte after the last file name character (or extension) should be a control character (value<32).

EXAMPLE: The following routine formats an FCB. Several results are shown along with their corresponding inputs.

```
        LD      C,134           ;FORMAT FILE NAME
        LD      DE,FCB
        LD      HL,FNAME
        CALL    5
        .
        .
        .
FNAME:          FILE1
FCB:            0FILE1      0
        .
FNAME:          A:FILE1
FCB:            1FILE1      0
        .
FNAME:          A:FILE1.TXT
FCB:            1FILE1   TXT0
        .
FNAME:          A:FILE1*.TXT
FCB:            1FILE1???TXT
```


Copyright 1983 OCCO inc.

REQUEST: UPDAT (UPDATE DIRECTORY)

INPUT PARAMETERS: C=135

OUTPUT PARAMETERS: none

FUNCTION: Write the current directory entry onto the disk.

COMMENTS: This call is normally used in conjunction with the SRCH and SRCHN calls. After a call to search for a file in the disk directory is executed, the directory entry address in MRS memory is returned to the user. The user may then modify the entry and use this call to update the entry on the disk. CAUTION should be used when executing this call as a change in the cluster list contained in the directory entry, could cause a linked cluster condition (2 files with the same disk space allocated to them). Note that if this call and the search for file calls are being used to change a file's name, the user should be sure to change the names in ALL the entries associated with that file.

EXAMPLE: Change a file's name where it is known that the file has only one entry associated with it.

```
        LD      C,17            ;SEARCH FOR FILE
        LD      DE,FCB
        CALL    5
        CP      A,-1            ;FILE FOUND?
        JR      Z,NOTFND        ;NO-ABORT ROUTINE
        INC     HL              ;POINT TO 1ST CHARACTER
        LD      (HL),'E'        ;CHANGE CHARACTER
        LD      C,135           ;UPDATE DIRECTORY
        CALL    5
        .
        .
        .
FCB:    DB      0,'FILE1      ',0

;NEW FILE NAME ON DISK = 'EILE1'
```


Copyright 1983 OCCO inc.

REQUEST: EXCMD (EXECUTE COMMAND LINE)

INPUT PARAMETERS: C=136

OUTPUT PARAMETERS: none

FUNCTION: MRS will execute the command line starting at memory location 80H as if the command line came from the console.

COMMENTS: The command line should be setup as if they were read in from the console using thee RDBUF call. If the command cannot be executed, CPU control will revert to the MRS user monitor.

EXAMPLE: Have MRS execute the program "NEXT".

```
        LD      C,136
        CALL    5

        ORG     80H
CMDLN:  DB      4,'NEXT',0
```


Copyright 1983 OCCO inc.

REQUEST: MULT

(MULTIPLY)

INPUT PARAMETERS: C=137
DE=multiplier
HL=multiplicand

OUTPUT PARAMETERS: DE=HL*DE (16 lsb's of product)

FUNCTION: Multiply the two input parameters together and return the least significant 16 bits to the user.

COMMENTS: A convenient call for 8 bit arithmetic. The two parameters are multiplied as if they were 16 bit values so both parameter registers should have their 8 msb's cleared. The multiplication is executed without regard for sign.

EXAMPLE: Multiply two numbers in memory and store the result.

```
        LD      C,135           ;MULTIPLY NUMBERS
        LD      DE,(NUM1)
        LD      HL,(NUM2)
        CALL    5
        LD      (RSULT),DE      ;STORE RESULT
        .
        .
        .
NUM1:   DW      12              ;MULTIPLIER
NUM2:   DW      7               ;MULTIPLICAND
RSULT:  DW      84              ;PRODUCT
```


Copyright 1983 OCCO inc.

REQUEST: DIVID (DIVIDE)

INPUT PARAMETERS: C=1338
DE=divisor
HL=dividend

OUTPUT PARAMETERS: DE=remainder(HL/DE)
HL=quotient(HL/DE)

FUNCTION: Divide the two parameters and return the result to the user.

COMMENTS: The two numbers are divided without regard to sign.

EXAMPLE: Divide two numbers in memory and store the result.

```
        LD      C,138           ;DIVIDE NUM1/NUM2
        LD      DE,(NUM2)
        LD      HL,(NUM1)
        CALL    5
        LD      (REM),DE        ;STORE REMAINDER
        LD      (QUOT),HL       ;STORE QUOTIENT
        .
        .
        .
NUM1:   DW      85              ;DIVIDEND
NUM2:   DW      3               ;DIVISOR
REM:    DW      1               ;REMAINDER
QUOT:   DW      28              ;QUOTIENT
```


Copyright 1983 OCCO inc.

REQUEST: HOMHD (HOME HEAD)

INPUT PARAMETERS: C=139
B=disk# (0=default disk,1=A...15=P)

OUTPUT PARAMETERS: none

FUNCTION: Home the head on the drive specified by B.

COMMENTS: This call is usually used just prior to removing a disk from the drive.

EXAMPLE: Home the drive head and tell the user to remove the disk.

```
        LD      C,139           ;HOME HEAD
        LD      B,0             ;ON CURRENT DRIVE
        CALL    5
        LD      C,9             ;WRITE MESSAGE TO CONSOLE
        LD      DE,MESG         ;TO TAKE OUT DISK
        CALL    5
        .
        .
        .
MESG:   DB      'REMOVE DISK FROM CURRENT DRIVE$'
```


Copyright 1983 OCCO inc.

REQUEST: REDIO (READ I/O STATUS)

INPUT PARAMETERS: C=140

OUTPUT PARAMETERS: A=I/O status byte

FUNCTION: Return the current MRS I/O status byte.

COMMENTS: The 2 lsb's of the status byte reflect the steering that MRS is giving to the output to the console. If a bit is set then the output is going to that device.

bit 0 = console output
bit 1 = printer output

EXAMPLE: Check for printer output on.

```
LD      C,140
CALL    5
BIT     1,A             ;PRINTER ON?
JR      Z,POFF          ;NO
JR      PON             ;YES
```


Copyright 1983 OCCO inc.

REQUEST: VERSN (VERSION NUMBER)

INPUT PARAMETERS: C=141

OUTPUT PARAMETERS: HL=MRS version number

FUNCTION: Return the current MRS version number to the user.

COMMENTS: This call is used to allow the user program to determine if the current MRS level will support the system calls that may be used.

EXAMPLE: Check that the MRS version will support the multiply and divide functions.

```
LD      C,141          ;GET VERSION #
CALL    5
LD      A,17H          ;1.7 & BELOW=NO MULT/DIVD
CP      A,L            ;EARLIER VERSION?
JR      NC,NOMULT      ;YES-NO MULTIPLICATION
.
.
.
```


Copyright 1983 OCCO inc.

REQUEST: SETIO (SET I/O STATUS)

INPUT PARAMETERS: C=142
E=new I/O status byte

OUTPUT PARMETERS: none

FUNCTION: Set the MRS I/O status byte.

COMMENTS: This call is often used in conjunction with the call REDIO. The 2 lsb's of register E determine the steering that MRS will give to the console output. If the bit is set, the corresponding device will be written to for subsequent output to the console.

EXAMPLE: Turn on the printer to follow the console output.

```
LD      C,142
LD      E,02H
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: WRDAT (WRITE DATE)

INPUT PARAMETERS: C=143
B=DAY
D=MONTH
E=YEAR (-1900)

OUTPUT PARAMETERS: none

FUNCTION: Set the system date.

COMMENTS: The day,time and year are in binary format. The year is the current year in this century.

EXAMPLE: Set the current date in MRS.

```
LD      C,143
LD      B,DAY
LD      D,MONTH
LD      E,YEAR
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: RDDAT (READ DATE)

INPUT PARAMETERS: C=144

OUTPUT PARAMETERS: A=DAY
D=MONTH
E=YEAR

FUNCTION: Read the current system date.

COMMENTS: The same remarks apply to these parameters as to the parameters in the WRDAT call.

EXAMPLE: Read the date and save it.

```
LD      C,144
CALL    5
LD      IY,DATE
LD      (IY+0),A        ;SAVE DAY
LD      (IY+1),D        ;SAVE MONTH
LD      (IY+2),E        ;SAVE YEAR
```


Copyright 1983 OCCO inc.

REQUEST: WRTIM (WRITE TIME)

INPUT PARAMETERS: C=145
B=SECOND
D=MINUTE
E=HOUR

OUTPUT PARMETERS: none

FUNCTION: Set the system time.

COMMENTS: All the time parameters are in binary format.

EXAMPLE: Set the system time.

```
LD      C,145
LD      IY,TIME
LD      B,(IY+0)        ;SECONDS
LD      D,(IY+1)        ;MINUTES
LD      E,(IY+2)        ;HOURS
CALL    5
```


Copyright 1983 OCCO inc.

REQUEST: RDTIM (READ TIME)

INPUT PARAMETERS: C=146

OUTPUT PARAMETERS: A=SECONDS
D=MINUTES
E=HOURS

FUNCTION: Return the system time.

COMMENTS: The returned parameters are in binary format.

EXAMPLE: Read and store the system time.

```
LD      C,146
CALL    5
LD      IY,TIME
LD      (IY+0),A        ;SAVE SECONDS
LD      (IY+1),D        ;SAVE MINUTES
LD      (IY+2),E        ;SAVE HOURS
```


Copyright 1983 OCCO inc.

REQUEST: SRBAT (SET/READ BATCH CODE)

INPUT PARAMETERS: C=147
E=new batch code

OUTPUT PARAMETERS: A=old batch code

FUNCTION: Allow the user access to the batch code which is NOT altered by MRS but is retained in MRS memory.

COMMENTS: This is a call used by the system SSP BATCH utility. The user should not ordinarily use this call however, if a user program desires to stop the BATCH mode, he can set the batch code to 0. A value other than 0 causes the SSP BATCH utility to be loaded into memory following the transfer of control to MRS at the end of the user program (see SSP BATCH).

EXAMPLE: Stop the BATCH utility be clearing the program return code.

```
LD      C,147          ;CLEAR PROGRAM RETURN CODE
LD      E,0
CALL    5
JP      0              ;END PROGRAM
```


Copyright 1983 OCCO inc.

REQUEST: NOT IMPLEMENTED

INPUT PARAMETERS: C=148 to C=168

COMMENTS: Calls within this range and all other unimplemented calls, have no meaning in MRS. They will, if invoked by a user program, issue a warning to the console which will contain the warning notice and the specific call number that was used. Expansions to MRS should be in range but be careful that any added functions do not interfere with other operating system calls which some applications software may use.

Copyright 1983 OCCO inc.

STANDARD REQUESTS

C reg	MNEMONIC	FUNCTION
0	RESET	Reset MRS and return control to O/S
1	RDCE	Read char. from cosnsole with echo to A
2	WRTC	Write character in E to console
3	READR	Read byte from reader
4	PUNCH	Write byte to punch
5	PRTC	Write character in E to printer
6	DIRIO	Direct I/O to console via E
7	RDIO	Return with I/O byte in A
8	WRIO	Set the I/O bbyte
9	WRBUF	Write char. string to console (DE)
10	RDBUF	Read char. string from console (DE)
11	CSTA	Test status of console receiver A
12	VERSN	Return MRS version number to user
13	DSKRST	Reset the entire disk system
14	DSKSEL	Select drive specified in E as current
15	OPEN	Open specified file (DE) if it exists
16	CLOSE	Close the specified file (DE)
17	SRCH	Search for 1st extent of file (DE)
18	SRCHN	Search for next extent fo file (DE)
19	DELET	Delete specified disk file (DE)
20	RDREC	Read sequential record in file (DE)
21	WRREC	Write sequential record in file (DE)
22	CREAT	Create fiile with specified name (DE)
23	RENAM	Rename specified file (DE)
24	LOGIN	Provide bit map in A of logged in disks
25	CURDSK	Provide number in A of default disk
26	BUFSET	Change disk I/O buffer address (DE)
27	ALOVCT	Provide information on current disk
28	WRTPRT	Write protect current disk
29	ROVCT	Provide bit map in A of read only disks
30	SETFLG	Set/reset write protect flag
31	GETDPB	Get the current disk's DPB address
32	USRCD	Read or set the user code
33	RDFIL	Read record using random access pointer
34	WRFIL	Write record using random access pointer
35	COMSIZ	Compute size of specified file (DE)
36	COMREC	Compute random record number
37	DSKRST	Reset all disk drives
38	NOTIMPL	Not implemented in MRS
39	NOTIMPL	Not implemented in MRS
40	WRZRO	Write zeroes to specified record (DE)

Copyright 1983 OCCO inc.

MRS ONLY REQUESTS

C reg	MNEMONIC	FUNCTION
128	RDC	Read character with NO echo
129	NOTIMPL	Not implemented in MRS
130	CNTRC	Set routine address for CONTROL-C
131	RDLOG	Read specified logical record from disk
132	WRLOG	Write specified logical record to disk
133	CLSTIO	Enable/disable cluster I/O
134	FORMAT	Format a file name into an FCB
135	UPDAT	Update current directory entry
136	EXCMD	Execute command linestrating at 80H
137	MULT	Multiply (DE=HL*DE)
138	DIVID	Divide DE=remain(HL/DE),HL=quotnt(HL/DE)
139	HOMHD	Home head of drive specified in B
140	REDIO	Return MRS I/O status byte in A
141	VERSN	Return MRS version number to user
142	SETIO	Set MRS I/O status byte as specified in E
143	WRDAT	Set current date in MRS
144	RDDAT	Read the current system date
145	WRTIM	Set the current time in MRS
146	RDTIM	Return the system time
147	SRBAT	Set/Read batch code
148-168	NOTIMPL	Not implemented in MRS

Copyright 1983 OCCO inc.

Z80 CARD

Foundation Computing

8/20/87

Dear Jim,

Enclosed is the modem and the Z80A system. I booted up the system and checked all operations before packing it. Everything checks out O.K.

There are two terminal emulator programs enclosed along with the systems disk. The T.E. on the Nashua disk is a bare-bones version that come with the original package. The terminal emulator on the Datalife disk is an enhanced version that works much better than the original.

You'll need to read the manual closely to get started. Where the manual is unclear or has omitted pertanent information, I've included some clarifying comments.

The manual gives some useful information on obtaining software. Just remember that this system is functionally equivalent and compatible with the Kaypro 2 computer (except that the display is 40 column instead of 80 column). So Kaypro 2 software sources are a good place to start.

If you have any problems or questions, let me know and I'll try to help.

Sincerely,

James A. Chanen