PLEASE NOTE THE FOLLOWING SPECIAL PRECAUTIONS WHEN USING DISKETTES

There are a few special precautions you must observe when handling diskettes and files to avoid destruction of data and programs through misuse or mishandling:

1. Whenever you remove a diskette from a drive and replace it with another diskette, REBOOT the CP/M system BEFORE PERFORMING ANY SUBSEQUENT OPERATIONS. A "warm start" is sufficient (control-C) to cause CP/M to recognize that the diskettes have changed. A reboot is not necessary, however, if the replaced diskette is "read-only" and data or programs will not be written to the diskette.

2. Do not turn the mainframe or disk drive power off with a diskette in the drive. Many controllers (such as the MDS 800 controller) will engage the head and turn on the write electronics momentarily, thus destroying a track of data.

3. Always store diskettes in their protective jackets when not in the diskette drive. Otherwise, dust will gather on the recording surface causing drive head wear and reduced media life.

4. Store the diskettes in normal work areas where temperatures are not extreme (within the 50-125 degrees F), and do not allow them to get near magnetic influences (such as large power supply transformers) or allow them to be exposed to direct sunlight for any extended period of time.

5. Provide adequate archives for your programs and data. Regular and organized backup techniques are essential for protection against media, hardware, software, or operator failures in any computing environment.
TABLE OF CONTENTS

1.0 Parts List
2.0 Minimum Hardware Configuration
3.0 How To Connect The Hardware
4.0 Bringing Up the System
   4.1 Connecting the Console
   4.2 System Console Communications
   4.3 Teletype
   4.4 RS232 Device
   4.5 Baud Rate Selection
   4.6 DIP pin layout
5.0 Bringing Up CP/M
   5.1 Bringing Up the System Without A Front Panel
   5.2 How To Insert A Diskette
6.0 Backing Up the System
   6.1 Altering the Software I/O Port Speed
   6.2 Serial I/O Port Speed Constants
7.0 Relocating Your System
   7.1 Operating System Components
   7.2 Regeneration Procedure
   7.3 Sample I/O Routines
   7.4 Common Load Offsets
8.0 Theory of Operation
   8.1 How CP/M Is Initiated
   8.2 Schematics
9.0 Source Code
   9.1 CBIOS
   9.2 Bootstrap
   9.3 Superboot
   9.4 Software Serial I/O Port
<table>
<thead>
<tr>
<th>QUANTITY</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dual Disk Drive</td>
</tr>
<tr>
<td>1</td>
<td>S-100 Bus Controller Card</td>
</tr>
<tr>
<td>1</td>
<td>40 Conductor Connector Cable</td>
</tr>
<tr>
<td>1</td>
<td>Software Diskette</td>
</tr>
<tr>
<td>1</td>
<td>16 Pin DIP Header</td>
</tr>
<tr>
<td>1</td>
<td>Documentation Manual</td>
</tr>
<tr>
<td>1</td>
<td>Warranty Card</td>
</tr>
<tr>
<td>1</td>
<td>CP/M Registration Card</td>
</tr>
<tr>
<td>1</td>
<td>CP/M Licensing Agreement</td>
</tr>
<tr>
<td>6</td>
<td>CP/M Documentation Manuals</td>
</tr>
<tr>
<td>1</td>
<td>BASIC-E Reference Manual</td>
</tr>
</tbody>
</table>
SECTION 2.0

MINIMUM HARDWARE CONFIGURATION

The MICROMATION Dual Disk Drive system runs on an S-100 bus, 8080 or Z-80 microcomputer with a minimum of 16K of RAM. To run BASIC-E at least 20K is needed. The memory must be contiguously addressed from locations 0 through 3FFF. Additional memory must avoid certain locations due to the memory on the controller board. The reserved locations are four pages of memory, F800 through FBFF.

A console communications device is also needed. A video terminal (CRT) such as the ADM-3 or Hazeltine 1500 or a hardcopy type such as the Decwriter II, teletype or the HyTerm II is adequate. The terminal must be able to communicate over a standard serial interface using RS232 conventions or a Teletype terminal (TTY) that uses a 20ma current loop.

Computers that do not have front panel switches or some other means of transferring control to a specified location (other than zero) will need a board that transfers control on a reset or power-on.

SECTION 3.0

HOW TO CONNECT THE HARDWARE

Be sure that all components of the computer system are unplugged before connecting the MICROMATION dual disk drive.

Place the controller card in an empty S-100 bus slot component side forward. The card must be firmly seated in the connector to make good electrical contact. The connector end of the 40 conductor cable extending from the MICROMATION disk drive unit should then be connected to the top row of pins on the controller board. A small number "1" is etched near the leftmost pin on the board. The side of the cable with the single red wire must align with this. In most computers the cable will extend toward the rear of the box.
Any board that executes a jump instruction on a reset or power-on, such as the MICROMATION JUMP-START (tm) board, should be disabled until the disk system has been tested (If you do not have front panel switches, you WILL need a JUMP-START board or equivalent to get started. See Section 5.1). The JUMP-START board will be a helpful addition to the system when it is up and running.

SECTION 4.0

BRINGING UP THE SYSTEM

Supplied with the MICROMATION disk system is a copy of CP/M, a microcomputer Operating System (OS). It provides a named file structure on diskettes and I/O routines for the system's peripheral devices. It includes system tools such as an assembler, text editor and dynamic debugger.

CP/M comes with six separate manuals describing its abilities and use. It is important that these manuals be read and studied.

This manual is a guide to the CP/M documentation and it provides a convenient summary for generating CP/M systems. It will not act as a substitute for a thorough reading of the CP/M documentation. The contents of this manual and the CP/M manuals should be read completely before attempting any action.

SECTION 4.1

SYSTEM CONSOLE COMMUNICATION

CP/M uses a two-way communications device called the System Console. Through it the user requests services from the operating system and the OS informs the user of its status. The console device is usually a Cathode Ray Tube (CRT) or a Teletype (TTY). In addition to the console device itself, two things are necessary. First is a serial I/O port that supports either the ITY interface or the RS232 terminal and the software routines to interface between this port and the CP/M operating system. Both are supplied with the MICROMATION system. The I/O port is implemented in an on-board PROM. The interface drivers are ready to run in the distributed version of the CP/M BIOS (See CP/M SYSTEM
The serial I/O port provides instant communication with the system. It is possible to start running right away. It also allows the use of the CP/M facilities to customize CP/M avoiding the laborious task of hand assembling and toggling in of I/O routines. It also avoids the undesirable practice of "hot patching" programs.

The primary purpose of the on-board software I/O port is to get the system running with a minimum of time and effort. It is not designed as a permanent replacement for hardware I/O support. Many hardware I/O boards provide more than one port and a wider range of communication disciplines than the on-board software port.

SECTION 4.2

CONNECTING THE CONSOLE

The MICROMATION controller board has a 16 pin DIP socket in the upper right corner for connecting the system console device. The console device MUST have either an RS232 or 20ma current loop interface. A 16 pin DIP Header (plug) is provided with the MICROMATION system. It should already be inserted in the DIP socket. Remove it before soldering to it! The connecting wires from the terminal must be soldered to the top of the pins on the plug. The pin configuration is shown in Figure 4.6. If the terminal is a TTY or other current loop device connect only the pins shown in Figure 4.4. If it is an RS232 compatible device connect only the pins shown in Figure 4.3. Some RS232 devices require that lines other than the three shown here be pulled high or grounded. Check the specific manufacturer's documentation before hooking up any additional lines.

Data is sent through the port serially with no parity. There is one start bit and two stop bits on each data byte. Assure that the terminal is set for this type of communication.

Once the soldering is finished, plug the DIP header into the socket on the controller board. The notches on both components must be aligned.
SECTION 4.5

BAUD RATE SELECTION

The serial data transmission speed of the software I/O port is regulated by a two byte constant held in memory in the CP/M CB IOS. It is set initially for 110 baud (10 cps), the speed of TTYs. Most CRTs can be set for this speed. If the terminal will operate at higher speeds it is possible to alter the speed constant. This should be changed only after the system is in an operational state. Once the system is up
the speed can be altered with the facilities of CP/M. A description of how to change the baud rate constant is given in Section 6.1.

SECTION 4.6

DIP PIN LAYOUT

SECTION 5.0

CP/M INITIATION PROCEDURE

When all of the hardware components are properly installed CP/M can be initiated and run. It is strongly suggested to make at least one backup copy of the system diskette immediately after determining that the system is functional and before ANY other processing. A program is included with the system for this purpose. The procedure for using it is described in Section 6.0. The following steps describe the exact procedure for starting CP/M.
A. All components must be interconnected. The 40 conductor cable should run from the disk controller to the disk drives. The console must be connected to the 16 pin DIP header.

B. All components must be plugged into a grounded, 115 vac circuit. Be sure that the console device is plugged in.

C. Turn on the power to the computer and the disk drives. Depress the "RESET" switch on the computer's front panel. The "STOP" switch, if the computer has one, should be depressed before the "RESET" to prevent the computer from executing random instructions before the bootstrap operation. Allowing the computer to process "garbage" instructions can cause it to write garbage on the diskette! Do not turn on the power while the diskette is in the drive as power transients can destroy data also.

D. Insert the system diskette in drive "A". Make sure the diskette is facing in the proper direction (See Figure 5.2). CP/M will ALWAYS bootstrap from drive "A".

E. Examine memory location F800. This is the beginning of the program "SUPERBOOT" in PROM. Verify that the first byte of this routine is a 31H. The system will now be ready to execute the procedure to bootstrap the operating system into main memory.

F. Start processor execution with the "RUN" switch and the "SUPERBOOT" routine will bring in the bootstrap program from track zero, sector one of drive "A". The bootstrap program will then read in the remainder of the OS from tracks zero and one.

The system bootstrap will take approximately three seconds. When the operation is finished, the CP/M Console Command Processor (CCP) will type the system prompt message to the console. It looks like this:

A>
The prompt message is printed whenever CP/M is idling and awaiting a command from the console operator.

If some combination of one or two other characters appear on the screen it may indicate a communications problem. Check the console device to assure that it is set for the proper baud rate and framing pattern. If there is no response from the terminal, check the manufacturer's documentation. Some RS232 terminals require a "Clear To Send" or other signal to be pulled high before they will respond to any external communications. Some devices can automatically transmit a line-feed following a carriage-return. If this option is present, it must be disabled.

Briefly test the CP/M functions at this point. Type the command "DIR" followed by a carriage-return and the operating system should respond by printing the diskette file directory. See the CP/M FACILITIES manual for a more detailed description of the CCP functions.

Test the resident command "TYPE" by typing:

   TYPE BOOT.ASM

The source file for the bootstrap program "BOOT" should be printed to the system console (long typeouts can be aborted by hitting any key on the console keyboard.).

The next test should be of a CP/M system transient program. Use the "STAT" transient for this. Type:

   STAT

The response should be:

   BYTES AVAILABLE: nnnK

To test the write function of the system type:

   SAVE 1 X.COM

This will build a small file on diskette by the "X.COM". The "DIR" command should show that the file has been added to the directory.

At this point the system is functioning correctly. Before attempting any programming tasks at least one system backup disk should be created.
SECTION 5.1

BRINGING UP THE SYSTEM WITHOUT A FRONT PANEL

A power-on/reset jump start board is necessary to bring up the MICROMATION system if the computer does not have a front panel. Not having a front panel reduces the debugging facilities available but the initiation procedure is simplified.

Set the jump address on the board for F800, the address of Superboot. Follow the procedures described in section 5.0 except for step 'E' which will be automatically performed by the jump start board.

SECTION 5.2

HOW TO INSERT A DISKETTE

For Memorex systems, insert the diskette in drive A, the lower drive, with the label facing up. Push the diskette firmly until it engages in the drive, and close the door of the drive.
SECTION 6.0

BACKING UP THE SYSTEM

The program MMCOPY copies the entire contents of a diskette from drive A onto a diskette on drive B. Place a blank diskette in drive B. Be sure that the write protect notch is absent or has a tab over it. This will enable the write mechanism of the MICROMATION drive. Type the following in response to the system prompt:

```
MMCOPY
```

To make more than one backup diskette, type:

```
MMCOPY R
```

This causes the MMCOPY program to repeat the copying operation. When the copy program requests a return, make sure the diskettes are inserted and type return. When the diskettes have been copied, respond with a control-c to the "TYPE RETURN" message (See the MMCOPY documentation for a complete discussion of this program).

One backup copy of the system should be stored in a protected location and kept only in the event that all other diskettes are erased. Remember, if the last system diskette is accidently erased it will cost $25.00 to replace it. BACK IT UP!

SECTION 6.1

ALTERING THE SOFTWARE I/O PORT SPEED

If the system console device will run at a higher speed than the preset 110 baud rate, the speed constant held in reserved memory locations FA71 and FA72 can be altered. For baud rates of 110 or faster only the low order byte is significant. The high order byte is always set to zero. The single low order byte can be set to any new speed with the Dynamic Debugging Tool, DDT. After initiating DDT, use the 'set' facility to insert the proper speed constant in the low order location. The front panel can also be used. The speed constants are shown in Table 6.2.
Immediately after setting the constant, communications to and from the console will become garbled until the baud rate on the console device is changed.

Altering the speed constant with DDT is temporary. It will only last until the next "COLD" bootstrap operation. A cold boot will bring in a fresh copy of the CBIOS from diskette. It will contain the old speed constant for 110 and the system will instantly revert. A permanent change to the I/O port speed can be done when relocating CP/M.

SECTION 6.2

SERIAL I/O PORT SPEED CONSTANTS

<table>
<thead>
<tr>
<th>BAUD RATE</th>
<th>DECIMAL</th>
<th>HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>171</td>
<td>AB</td>
</tr>
<tr>
<td>150</td>
<td>125</td>
<td>7D</td>
</tr>
<tr>
<td>300</td>
<td>62</td>
<td>3E</td>
</tr>
<tr>
<td>600</td>
<td>31</td>
<td>1F</td>
</tr>
<tr>
<td>1200</td>
<td>15</td>
<td>0F</td>
</tr>
<tr>
<td>2400</td>
<td>7</td>
<td>07</td>
</tr>
</tbody>
</table>

SECTION 7.0

RELOCATING YOUR SYSTEM

When the CP/M disk system is up and running it is possible to generate a system that will utilize all available RAM. To run BASIC-E or CBASIC a system running in at least 20K of memory is needed.

While adjusting the size of the operating system the CBIOS console I/O routines can be replaced. The new I/O routines can communicate through the normal I/O ports and hardware I/O support board(s). A driver routine to allow CP/M to output to a printer can be installed.
The procedures for generating, relocating and customizing the operating system are thoroughly described in the CP/M documentation manual:

CP/M SYSTEM ALTERATION GUIDE

The following section is a step-by-step summary of how to relocate and customize the system. This manual is not as detailed as the SYSTEM ALTERATION GUIDE. It is not a substitute but an aid and summary for the CP/M manual. The process of generating a new system is not complex but it can be a confusing procedure the first few times it is attempted. It is suggested that both this and the CP/M manuals be studied before building a custom system.

SECTION 7.1

OPERATING SYSTEM COMPONENTS

CP/M is composed of resident and transient programs. The transient programs need no modifications because they adjust themselves to the size of the current operating system. The resident components of the operating system must be modified for different sizes. They are:

CONSOLE COMMAND PROCESSOR
BASIC DISK OPERATING SYSTEM
BASIC INPUT OUTPUT SYSTEM
BOOTSTRAP PROGRAM

A detailed discussion of the organization of resident CP/M components is in the manual:

CP/M INTERFACE GUIDE

When I/O drivers are added to the Basic Input Output System (BIOS) a Customized BIOS or CBIOS is created. This CBIOS must be assembled for the desired system size and combined with the other portions of the OS including the bootstrap program. A standard system program, MOVCPM, will regenerate a new version of the operating system of any desired size without the customized portions and the bootstrap.
SECTION 7.2

REGENERATION PROCEDURE

Create the operating system (CCP and BDOS) for the size desired. The CBIOS and Bootstrap programs are distributed in source form. They need to be modified and reassembled separately. The three components are then gathered in the transient program area with DDT and saved on diskette as a COM type file.

Use the MOVCPM program to create a new copy of the operating system of the desired size. Save it as an ordinary COM file by typing:

```
SAVE 32 CPMnnK.COM
```

Where 'nn' is the size of the new system. Use the PIP program to make a copy of the BIOS source that is distributed with the system. Call it 'CBIOSnnK.ASM'. Copy the bootstrap 'ASM' file with PIP also. Name it 'BOOTnnK.ASM'.

Make the following modifications to the source files of the CBIOS and Bootstrap. Use the CP/M Text Editor (ED).

The manual:

**ED: A CONTEXT EDITOR FOR THE CP/M DISK SYSTEM**

gives a thorough description of the use of this program.

Enter the Editor with the name of the CBIOS source copy. Alter the 'MSIZE' variable at the start of the program so that it indicates the proper size of the system. Put in comments describing any changes made in the CBIOS. Put the date of the change at the beginning of the code. Insert the I/O drivers for peripheral access after the names:

```
CONIN:
CONOUT:
CONST:
```

A detailed description of the purpose of each routine can be found on page 15 of the SYSTEM ALTERATION GUIDE.

Be sure the I/O routines do not force the size of the CBIOS out of the allotted space.

For a list device, enter the driver routine after the name:
LIST:

Routines for a paper tape reader or punch can be added after the names:

READER:
PUNCH:

Unused routines should be terminated with a 'RET' instruction. If the hardware I/O board has UART chips that need programming, those procedures should be installed in the cold boot portion of the CBIOS marked with the comment:

;PLACE UART INITIALIZATION RTNS HERE

Sample I/O routines are shown in Section 7.3.

When all modifications have been made to the CBIOS, exit from the editor and assemble the new CBIOS. Detailed instructions on the CP/M Assembler are in the manual:

CP/M ASSEMBLER (ASM)

In addition to the CBIOS the bootstrap program must be reassembled for the new memory size. Using the editor, change the 'MSIZE' variable at the start of the bootstrap program. Exit the editor and reassemble the bootstrap.

CP/M can now be created out of its components: the relocated operating system (CCP, BDOS), the customized CBIOS, and the bootstrap. Uniting all of these parts is done with DDT. Start the DDT program and read in the new CP/M by typing:

DDT CPMnnK.COM

DDT will respond with its logon message followed by the next available address and the contents of the program counter:

DDT VERS 1.3
NEXT PC
2100 0100

Now insert the name of the CBIOS file by typing:

ICBIOSnnK.HEX

This prepares DDT to read the CBIOS file. DDT normally reads programs into the memory locations for which they have been assembled. The operating system must be built in the TPA rather than the location where the OS will reside when it is running. They can be placed in the proper location by reading the files in with an "offset". The offset for the CBIOS is calculated from the size of the new system. The offset and calculating it is fully explained on pages 6-7.
the SYSTEM ALTERATION GUIDE.

The lowest page of memory is reserved for system communications and usable memory begins at location 100H. The SYSGEN program will occupy the 800H bytes ranging from 100H to 8FFH. The new operating system must be placed starting at location 900H where the SYSGEN program expects to find it.

Section 7.4 shows a chart of common offsets. For example, use the offset of A080 for a 32K system. To read the CBIOS with the 32K offset, type:

RA080

This causes the CBIOS to be properly inserted in relation to the BDOS forming the correct operating system configuration.

Insert the bootstrap program next. The bootstrap's location does not change from system to system, it is always loaded with the same offset and will always occupy the first sector on diskette. It must occupy the lowest portion of memory beyond 900H. The bootstrap is "org'ed" at location zero and must be loaded by DDT at an address 900H bytes away from its normal load address. This is done by specifying a 900H byte offset on the DDT "R" (READ) command:

IBOOTnnK.HEX
R900

The new customized CP/M is now properly organized in memory. Type a control-c to return to the monitor and type:

SAVE 32 CPMnnK.COM

This places a copy of the customized system onto the diskette under the name specified. The system must be "sysgen'ed" onto the first two tracks of a diskette for bootstrapping. The first two tracks are what the bootstrap program reads.

The program 'SYSGEN' is used for accessing the operating system tracks. SYSGEN performs two vital tasks. It reads a copy of the operating system off of tracks zero and one and places it into memory starting at location 900H in the Transient Program Area, and two, it will take any copy of the operating system that is already at location 900H and place it on the first two tracks of the specified diskette.

Use DDT to get the copy of the new system into the TPA, then use the second function of SYSGEN to place it on any diskette. The sequence is as follows:
When DDT finishes loading the new OS and types the prompt character, enter a control-c to return to the monitor. Place the desired diskette in drive B and call SYSGEN to place the operating system on it. When SYSGEN types:

GET SYSTEM (Y/N)?

Type a 'N'; the system is already present. SYSGEN will then request:

PUT SYSTEM (Y/N)?

Respond with a 'Y' and the SYSGEN program will place the new OS onto the first two tracks of the diskette in drive B from the image of the operating system in memory.

When the SYSGEN is finished, the diskette in drive B is ready for rebooting. Remove the diskette from drive A and replace it with the one in B. Remember, in order to read in the ENTIRE new copy of CP/M, execute a cold-start (RESET) procedure. The control-c operation only performs a warm boot and will not read in the CBIOS.

SECTION 7.3

SAMPLE CBIOS I/O ROUTINES

CONST:

| IN | 0 | ;GET PORT STATUS |
| ANI | 1H | ;IS A CHAR THERE? |
| JNZ | WASTHERE | ;YES, SO JUMP |
| XRA | A | ;NO, SO CLEAR FLAGS AND ACCUM |
| RET | ;ALL DONE. EXIT |

WASTHERE:

| MVI | A, OFFH | ;INDICATE THAT SOMETHING |
| RET | ; WAS THERE THEN EXIT |

CONIN:

| IN | 0 | ;GET PORT STATUS |
| ANI | 1H | ;IS A CHAR THERE? |
| JZ | CONIN | ;IF NOTHING THERE, TRY AGAIN |
| IN | 1 | ;GET THE INCOMING DATA BYTE |
| ANI | 07FH | ;CLEAR THE PARITY BIT |
| RET | ;ALL DONE. EXIT |

CONOUT:

| IN | 0 | ;GET PORT STATUS |
ANI 2H
JZ CONOUT ; IS OUTPUT BUFFER CLEAR?
MOV A,C ; PUT OUTGOING DATA IN ACCUM
OUT 1 ; WRITE IT OUT
RET ; ALL DONE. EXIT

LIST:
IN 2 ; GET PORT STATUS
ANI 2H ; IS OUTPUT BUFFER CLEAR?
JZ LIST ; NO, SO TRY AGAIN
MOV A,C ; PUT OUTGOING DATA IN ACCUM
OUT 3 ; WRITE IT OUT
RET ; ALL DONE. EXIT

PUNCH:
READER:
RET ; PUNCH AND READER ARE NOT USED.

SECTION 7.4

< COMMON OFFSETS

<table>
<thead>
<tr>
<th>SIZE IN K</th>
<th>OFFSET VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>E080</td>
</tr>
<tr>
<td>24</td>
<td>C080</td>
</tr>
<tr>
<td>32</td>
<td>A080</td>
</tr>
<tr>
<td>40</td>
<td>8080</td>
</tr>
<tr>
<td>48</td>
<td>6080</td>
</tr>
<tr>
<td>56</td>
<td>4080</td>
</tr>
<tr>
<td>64</td>
<td>2080</td>
</tr>
</tbody>
</table>

19
The MICROMATION disks are controlled by an S-100 bus compatible controller. The controller is managed by software in two pages (512 bytes) of on-board PROM. Data transferred is buffered in one page (256 bytes) of on-board RAM. An additional address page is reserved for use by the controller for communications. The memory is addressed as follows:

<table>
<thead>
<tr>
<th>PROM</th>
<th>F800</th>
<th>F9FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>FA00</td>
<td>FAFF</td>
</tr>
<tr>
<td>I/O LOCATIONS</td>
<td>FB00</td>
<td>FBFF</td>
</tr>
</tbody>
</table>

The MICROMATION controller transfers information to and from diskette whenever one of two reserved memory locations is accessed. When the 'MARKPORT' byte is read the controller reads a sectormark from the diskette. When the same byte is written to the sectormark is written to diskette. A second 'pseudoport' is called 'DATAPORT'. When it is accessed it causes transfer of a single byte of data. The pseudoport can be considered as an output port to diskette. For example, if the instruction:

```
LDAX DATAPORT
```

is executed, the transfer of data is to the accumulator from the diskette rather than from the memory byte itself. Conversely, if this is executed:

```
STAX DATAPORT
```

The byte in the accumulator is written to the current disk location. Each sector of data is arrayed on diskette in the following format:
The status of the disk controller can be read into the accumulator in the same manner as data is transferred. By reading the 'statusport' location, eight bits of information are placed in the accumulator.

\[
\begin{array}{c|c}
\text{D7} & \text{READY} \\
\text{D6} & \text{SEEK} \\
\text{D5} & \text{HEAD} \\
\text{D4} & \text{INDEX} \\
\text{D3} & \text{SECTOR} \\
\text{D2} & \text{WRITE} \\
\text{D1} & \text{SERIAL} \\
\text{DO} & \text{TRACK} \\
\end{array}
\]

The controller is given instructions by writing to the 'CONTROLPORT' memory location. The eight control bits are:

\[
\begin{array}{c|c}
\text{D7} & \text{UNIT} \\
\text{D6} & \text{SELECT} \\
\text{D5} & \text{SELECT RESTORE} \\
\text{D4} & \text{A} \\
\text{D3} & \text{B} \\
\text{D2} & \text{DIR} \\
\text{D1} & \text{STEP} \\
\text{DO} & \text{HEAD} \\
\end{array}
\]

SECTION 8.1

HOW CP/M IS INITIATED

A small program, called 'SUPERBOOT' is burned into PROM at the reserved memory location F800. This program has the single function of reading in a single sector of data from track 0 sector 1 of drive A. It places the 128 bytes of data at location zero in main memory. Execution is then transferred to location zero. The 128 byte program that SUPERBOOT loads is the cold bootstrap loader for CP/M. SUPERBOOT is the same for any MICROMATION version of CP/M. When initiating CP/M for the first time, either a manual operation or a power-on/reset triggered circuit must jump to location F800 where the SUPERBOOT program resides.
This program is a generalized full-disk copy program that is designed to run in a CP/M environment. MMCOPY will copy the entire contents of a diskette on drive A to a diskette on drive B.

The program is invoked by typing the transient name with two optional parameters. For example:

A-MMCOPY RS

If the optional 'R' parameter is specified the program will repeat execution indefinitely or until a control-c (warm boot) is entered from the console in response to the mount message. The mount message is issued before every copy and is of the form:

SOURCE ON A, DESTINATION ON B, THEN RETURN

It gives the operator a chance to change either one or both of the diskettes. After the diskette has been copied or when a control-c is detected MMCOPY will issue a reboot message giving the operator the opportunity to mount a system diskette in drive A.

If the optional 'S' parameter is entered anywhere on the command-line, the copy program will stop copying when it encounters a full track of 'E5's. When a diskette is initialized it is padded with the hexadecimal byte configuration of 'E5's. The 'S' parameter will thus allow a diskette with only a few tracks used to be copied in significantly less time than if the entire 77 tracks of unused data area were copied.

All data copied is automatically verified on disk reads and writes. If an error is detected the entire track (26 sectors) will be recopied and a message will be printed indicating the hex address of the track and sector in error. If the error persists, MMCOPY will retry the track for 10 times. After 10 unsuccessful retries a 'PERMANENT' message will be printed and the program will continue, ignoring the bad data.
MEMTEST is a program designed to give your RAM memory an extensive read and write test. It will record all errors found while running on the system console. It is designed to test only RAM memory and will not test disk I/O or disk DMA. Other programs are available to test those functions.

The program begins by requesting three addresses from the user that must be entered in hexadecimal form. Leading zeros are required. It will request a starting address, a test length and a test increment. The program will begin at the starting address and perform three tests (called phases) in a single block of memory that is the size specified by 'increment'. At the end of these three tests the starting address is increased to the next increment and the phases are repeated. This continues until enough increments have been tested to equal the test length. For example, if you specify a starting address of 4000 (remember, this is in hex), a test length of 4000 and an increment of 1000, the program will start at 4000 (16K) and test through 4FFF outputing error statistics at the end. It will then repeat the procedure starting at 5000 and testing through 5FFF. This repeats a total of four times or until it has tested 4000 (16K) bytes.

In order to thoroughly test a memory board, we recommend two tests. The first time through, the increment should be equal to the amount of memory addressed by one bank of RAM chips (this is 4K on most 16K boards). The second test should be run with an increment equal to the total board memory size. Thus on a 16K system with the test board strapped for 4000H through 7FFFH the test specifications would be:

<table>
<thead>
<tr>
<th>TEST ONE</th>
<th>TEST TWO</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEG. TEST LOCATION:</td>
<td>4000</td>
</tr>
<tr>
<td>TOTAL TEST LENGTH:</td>
<td>4000</td>
</tr>
<tr>
<td>TEST INCREMENT:</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>4000</td>
</tr>
</tbody>
</table>

Note that this is a very thorough test. Test one will take close to six hours to run to completion and Test two will take around 24 hours to execute!

The actual test consists of three phases. For Phase One the test area is written with a bit pattern and then examined to see if the pattern is still there. The program repeats this test 256 times checking all possible bit patterns.
In Phase Two the test area is initially filled with zeros. A byte containing a single '1' bit is then written to the first location of the test area. The entire increment is checked to see if it is still zeros. The program repeats this test routine through all eight single bit patterns. It will then write the test byte in the second location in the test increment and repeat the above loop. It will thus 'walk' the eight bit patterns through every byte in the test area.

The Phase Three test procedure is identical to Phase Two except that both the testing field and the walking bit pattern are complemented. All of memory is filled with FF's and the bit patterns that are walked are the eight patterns containing a single '0' bit.

Upon completion of the three phases on each test increment, the program prints a table that consists of a row of eight four-digit hex numbers that are a count of the total number of errors found during the three phases. The left most number corresponds to the most significant bit (7) of the chip and right most to the least significant bit (0) of the chip.

One row is printed for each 0400H bytes tested. If a board containing 1K chips was tested, each row corresponds to one block of chips and each number in the row to a specific chip. However, if the board contained 4K chips then the first four rows correspond to one block of chips and the total errors attributable to a given chip would be the sum of the four numbers in the individual column (of four rows) belonging to that bank.

MEMTEST was originally designed as a stand-alone memory test program but has now been upgraded to run in a CP/M environment. Therefore, care must be taken not to give the memory test program addresses that will cause it to overlay itself or the BDOS (operating system) with any of the test patterns. The minimum beginning test location is 1000H (4K) and the total test length should never extend into the CP/M BDOS.

The most convenient method of operation is to strap your memory so that the board to be tested has addresses that are contained completely within the TPA (Transient Program Area). A second method is to strap the board to be tested with addresses completely ABOVE the operating system. For example, if the system contained two 16K RAM boards a 16K version of CP/M could be used and the board to be tested should be strapped for 4000H (16K) through 7FFFH (32K-1).

If neither of the above options are possible on your system it may be possible to run the test on one 4K block at a time and then restrap the board so all blocks can be tested.
The MEMTEST program will ask during the setup procedure whether you want detail error information by printing:

**RECORD EACH ERROR ON CONSOLE? (Y OR N)**

If you respond with a 'Y', each time a byte is found to be in error a line will be printed at the system console in the form:

```
A= aa aa GB= gg bb W= ww ww
```

Where aa aa is the address in memory in hexadecimal form of the bad byte. Where gg is what pattern the program expected to find. Where bb is what the program found. Where ww ww is the address currently containing the walking byte.

During setup, the MEMTEST program will ask:

**REPEAT TEST? (Y OR N)**

If the response is 'Y' the entire test procedure will repeat indefinitely.
UPDATED 1/2/78 TO TO RUN IN PROM AT F800

;COMBINED ROUTINES TO BE INCLUDED IN DISK CONTROL

FA00 =

SCRATCH: EQU 0F800-H+200H

; DRIVERS FOR MEMOREX DRIVE

FA00 =

SCBATCH: EQU 0F800-H+200H

; MASK EQ UATES

00FE = ADDRESSMARK EQU 0FEH
0040 = SEEKDONEMASK EQU 40H
0002 = INMASK EQU 02H
00FD = OUTMASK EQU 0FDH
0080 = READYMASK EQU 80H
0004 = HOMEMASK EQU 04H
020 = HEADMASK EQU 20H
0010 = AMASK EQU 10H
0008 = BMASK EQU 08H

0023 = HEADSETTLE EQU 35D
000A = STEPSETTLE EQU 10D
0006 = STEPDELAY EQU 6D

; BOOTSTRAP:
; THIS ROUTINE READS TRACK 0, SECTOR 1 INTO MEMORY
; AT LOCATION ZERO AND THEN JUMPS TO ZERO

FA70 =

STACK EQU SCRATCH+70H

0000 =

COLDBOOT EQU 0

F800 3170FA
F803 0EO0
F805 CD8AF8
F808 CD22F8
F80B C200F8
F80E 0EO1
?810 CD56F9
F813 010000
F816 CD64F9
F819 CDC1F8
F81C C200F8
F81F C30000

LXI SP,STACK ;SET STACK-POINTER-TO BUFFER
MVI C,0 ;SELECT DRIVE A
CALL SELDSK
CALL HOME
JNZ BOOTSTRAP ;LOOP IF DRIVE NOT READY
MVI C,1
CALL SETSEC ;SET SECTOR ONE
LXI B,COLDBOOT ;SET DMA ADDRESS
CALL SETDMA ;AT ZERO
CALL DISKREAD
JNZ BOOTSTRAP ;LOOP IF ERROR
JMP COLDBOOT ;JUMP TO BOOT
HOME:
CALL DISKREADY ;IS DEVICE READY?
RNC . ;IF CARRY SET, THEN DISK
LXI H,TRACK ;POINT H-L TO TRACKBUFFER

ATHOME:
CALL STEPIN ;STEP AWAY FROM HOME
LDAX D ;READ STATUS
RAR ;CHECK TRACK ZERO BIT
JC ATHOME ;CONTINUE STEPPING IN TILL NOT
GOHOME:
CALL STEPOUT ;GO TOWARDS HOME
LDAX D ;CHECK STATUS
RAR ;CHECK TRACK ZERO BIT
JNC GOHOME ;LOOP UNTIL AT HOME

INITIALIZE:
LXI H,ADDRPTR
MVI M,ADDRESSMARK

FILLLOOP:
INR L ;BUMP ADDRESS BUFFER PTr
MVI M,0 ;FILL BUFFER WITH ZERO
JNZ FILLLOOP
RET

SETTRK:
XRA A ;GET A ZERO
ORA C ;IS TRACK-0?
RM ;IF YES, RETURN
MVI A,76D ;COMPARE TO LAST TRACK
SUB C ;CHECK IF TRACK GREATER
RC ;POINT TO PRESENT TRACK
CALL DISKREADY ;CHECK READY
RNC ;RETURN IF NOT

STEPLOOP:
LXI H,TRACK ;POINT TO PRESENT TRACK
MOV A,M ;GET PRESENT TRACK
CMP C ;COMPARE TO DESIRED TRACK
JZ DONESTEP ;ON CORRECT TRACK
CALL STEPHEAD ;CARRY SET TO INDICATE
JMP STEPLOOP ;GO AROUND AGAIN

STEPHEAD:
JC STEPIN ;GO INWARDS IF CARRY SF

STEPOUT:
LDA CONTROLBYTE ;CHECK DRIVE SELECT /
DCR M ;DECREMENT TRACK BUFFER
ANI OUTMASK ;SET D1 FOR DIRECTION
JMP DOSTEP ;EXECUTE STEP

STEPIN:
LDA CONTROLBYTE ;INCREMENT TRACK REGISTER
INR M ;SET IN CODE

DOSTEP:
STAX D ;OUTPUT DIRECTION
INR A ;SET STEPA BIT
STAX D ;OUTPUT STEP
DCR A ;CLEAR STEP BIT
STAX D ;CLEAR STEP BIT ON POR
MVI B, STEPDELAY ;SET UP DELAY
CALL DELAY ;DELAY FOR STEP TIME
DONESTEP:
MVI B,STEPSETTLE ;SET UP HEAD SETTLING
CALL DELAY
RET

DISKREADY:
CALL HEADLOAD
; CAUTION IT IS ASSUMED THAT HEADLOAD SETS D,E TO DISKFUNCTION
XRA A ;CLEAR THE ZERO FLAG
LDAX D ;GET FUNCTION BYTE FROM
RLC ;READY BIT SHIFTED INTO CARRY
RC ;CARRY, ZERO SET
INR A ;CLEAR ZERO SET
RET ;DRIVE NOT READY

SELDISK:
CALL HEADLOAD
;GET ZEROS
XRA A ;ZERO=1 IF DRIVE A, ZI
ADD C
JZ SELECTA

SELECTA:
MVI A,BMASK ;GET SELECT MASK FOR 1
JMP DOSELECT

DOSELECT:
MVI A,AMASK ;GET SELECT MASK FOR 2
STA CONTROLBYTE ;SET UP DRIVE STATUS
STA DISKFUNCTION ;SEND TO CONTROLLER
XRA A ;SET ZERO FLAG
JMP CALDELAY ;CALL DELAY FOR HEADLOAD

DELAY:
MVI L,31 ;# OF MILISFCS DELAY

DELAYLP:
LDA DATAPORT ;THIS INSTRUCTION CAN
D,DISKFUNCTION ;A 32 MICRO-SECOND DR
;SET UP DRIVE STATUS
JNZ DELAY
L,31 ; IF THE HEAD IS LOADED

HEADLOAD:
LXI D,DISKFUNCTION
LDAX D ;READ DISK STATUS
ANI HEADMASK ;CHECK FOR HEAD LOAD?
INX D ;POINT TO HEADLOAD
LDAX D ;STROBE HEADLOAD COUNTER
DCX D ;SET D,E TO DISKFUNCTION, FOR

CALLDELAY:
MVI B,HEADSETTLE ;SET UP HEADSETTLING
CZ DELAY ;LET HEAD SETTLE
XRA A ;SET ZERO FLAG
RET ;FOR RETURN
; MAP OF SCRATCH AREA

FA6F = BOOTSTACK  EQU SCRATCH+5FH
FA70 = CONTROLBYTE  EQU SCRATCH+70H
FA71 = SPEED  EQU SCRATCH+71H
FA73 = RETRYCOUNT  EQU SCRATCH+73H
FA74 = DMAADDR  EQU SCRATCH+74H
FA76 = ADDRPTR  EQU SCRATCH+76H
FA7D = DATAPTR  EQU SCRATCH+7DH
FAFD = LASTDATA  EQU SCRATCH+0FDH
007D = DATABYTE  EQU 7DH
FA79 = SECTOR  EQU SCRATCH+79H
FA77 = TRACK  EQU SCRATCH+77H
FB00 = DATAPORT  EQU SCRATCH+100H
FB01 = MARKPORT  EQU SCRATCH+101H
FB02 = DISKFUNCTION  EQU SCRATCH+102H
FB03 = LOADPORT  EQU SCRATCH+103H
FB03 = SERIALROUTPORT  EQU SCRATCH+103H
FAFE = CRCBUFFER  EQU SCRATCH+0FEH
84BF = RESIDUE  EQU 84BFH

; DISKREAD:
F8C1 0E00  MVI C,0 ;SET READ FLAG
F8C3 CDFF8  CALL ENTRY ;EXECUTE READ
F8C6 C0  RNZ ;RETURN IF ERROR
F8C7 2D  DCR L ;POINT TO CRC
F8C8 EB  XCHG ;MOVE LAST BYTE ADDRESS TO DE
F8C9 217DFA  LXI H,DATAPTR ;POINT TO ADDR OF DATA MARK
F8CC CD70F9  CALL CRCÆCH ;COMPUTE CRC
F8CF 78  MOV A,B ;MOVE HIGH RESIDUE TO ACC
F8D0 B1  ORA C ;COMPARE TO C
F8D1 C0  RNZ ;CRC ERROR IF B,C NOT ZERO

; DATAXFER:
F8D2 0680  MVI B,128 ;SET BYTE COUNTER
F8D4 117DFA  LXI D,DATAPTR ;POINT TO DATA MARK
F8D7 2A74FA  LHLD DMAADDR ;POINT H,L TO DESTINATION

; XFERLOOP:
F8DA 13  INX D ;POINT TO NEXT BYTE IN BUFFER
F8DB 1A  LDAX D ;GET BYTE FROM BUFFER
F8DC 77  MOV M,A ;STORE BYTE IN MAIN MEMORY
F8DD 23  INX H ;POINT TO NEXT BYTE IN MEMORY
F8DE 05  DCR B ;HIT BYTE COUNTER
F8DF C2DAF8  JNZ XFERLOOP ;GO AROUND FOR MORE
F8E2 C9  RET ;ZERO SET TO INDICATE NO ERROR

; DISKWRITE:
F8E3 2A74FA  LHLD DMAADDR ;POINT TO DATA IN MAIN MEMORY
F8E6 EB  XCHG ;MOVE ADDRESS TO DE
F8E7 217EFA  LXI H,DATAPTR+1 ;POINT TO DATA BUFFER

; LOADLOOP:
F8EA 1A  LDAX D ;GET BYTE FROM MEMORY
F8EB 77  MOV M,A ;MOVE INTO BUFFER
F8EC 13  INX D ;NEXT BYTE IN MEMORY
F8ED 2C  INR L ;NEXT BUFFER BYTE
F8EE C2EF8  JNZ LOADLOOP ;END OF BUFFER?
F8F1 117DF8  LXI H,LASTDATA ;POINT TO LAST DAT BYTE
F8F4 2E7D  MVI L,DATABYTE ;LOAD LOW ORDER ADDRES OF
ENTRY: 
CALL DISKREADY ;CHECK FOR HEAD LOADED
RNZ ;DISK NOT READY
DI ;DISABLE INTERRUPTS TO PROTECT READ
CALL READWRITE ;EXECUTE READ OR WRITE
EI ;ENABLE INTERRUPTS FOLLOWING READ/
RC ;RETURN IF NO ERROR
MVI A,4 ;WRONG SECTOR HEADER READ?
CMP B ;B CONTAINS POINTER WHERE ERROR OCCURRED
JZ ENTRY ;RETRY IF WRONG SECTOR
RET ;RETURN WITH NO ZERO TO INDICATE 1

READWRITE:
LXI H, ADDRPRTR ;POINT TO ADDR MRK
LXI D, MARKPORT ;POINT TO PORT F
MVI B,6 ;SET BYTE COUNTER

ADDRMARKLOOP:
LDAX D ;READ MARK
CMP M ;ADDRESS MARK?
JNZ ADDRMARKLOOP ;IF NOT TRY AGAIN

ADDRESSHEADER:
INX H ;LOOK AT NEXT BYTE IN HEADER
LDA D ;READ NEXT BYTE FROM DISK
CMP M ;RIGHT DATA READ?
JNZ ADDRESSHEADER ;TRY AGAIN IF NOT DONE
DCR B ;HIT BYTE COUNTER

GAPLOOP:
LDAX D ;READ BYTE OF GAP
DCR B ;HIT BYTE COUNTER
JNZ GAPLOOP ;RETURN IF NOT LAST GAP BYTE
MOV A,C ;CHECK READ/WRITE FLAG
ORA A ;FLAG = 0?
ORA A ;CHECK READ/WRITE FLAG
XRA A ;SET UP TO WRITE ZEROS

ZEROWRITE:
STAX D ;WRITE A ZERO DATA BYTE
DCR C ;LAST BYTE
JNZ ZEROWRITE ;GO AROUND FOR M

WITEDATALOOP:
INX D ;POINT TO MARKPORT
INX H ;POINT TO DATA MARK
MOV A,M ;GET DATA MARK
STAX D ;WRITE DATA MARK
DCX D ;POINT TO DATAPOINT
INX H ;POINT TO DATA
MOV A,M ;GET DATA BYTE
STAX D ;WRITE DATA TO DISK
DCR L ;POINT TO NEXT BYTE
JNZ WITEDATALOOP ;LOOP IF NOT LAST BYTE
XRA A ;CLEAR ACC, SET ZERO
STAX D ;WRITE ZERO
RET ;FINISHED
READSECTOR:
F946 1A   LDAX D ;READ PASTA.CRAP IN GAP
F947 1A   LDAX D ;DITTO
F948 1A   LDAX D
F949 13   INX D ;POINT TO MARKPORT
F94A 23   INX H ;POINT TO DATA MARK
F94B 1A   LDAX D ;READ DATA MARK
F94C BE   CMP M ;COMPARE
F94D C0   RNZ ;RETURN WITH ERROR IF NOT
F94E 1B   DCX D ;POINT TO DATAPORT

READDATALOOP:
F94F 2C   INR L ;POINT TO NEXT BYTE IN BUFFER
F950 C8   RZ ;GET OUT IF LAST BYTE
F951 1A   LDAX D ;READ DATA BYTE
F952 77   MOV M,A ;STORE BYTE IN MEMORY
F953 C34FF9 JMP READDATALOOP ;GO AROUND FOR MORE;

SETSEC:
F956 2179FA LXI H,SECTOR ;POINT TO SECTOR BUFFER
F959 71   MOV M,C ;STORE REGISTER NUMBER FROM C R
F95A CD6AF9 CALL SETADDRCRC ;COMPUTE CRC OF HEADER
F95D 71   MOV M,C ;STORE FIRST CRC BYTE
F95E 23   INX H ;POINT TO NEXT BUFFER BYTE
F95F 70   MOV M,B ;STORE SECOND CRC BYTE
F960 23   INX H ;POINT TO NEXT BYTE
F961 36FB   MVI M,0FBH ;STORE DATA MARK
F963 C9   RET ;DONE

SETDMA:
F964 60   MOV H,B ;MOVE B,C PAIR TO H,L
F965 69   MOV L,C
F966 2274FA SHLD DMAADDR ;STORE ADDRESS IN BUFFER
F969 C9   RET
SETADDR CRC:

F96A 2176FA LXI H, ADDRPTR ; STARTING ADDRESS IN H
F96D 1174FA LXI D, SECTOR+1 ; ENDING ADDRESS IN D, E

CREECH: ; ROUTINE TO COMPUTE CRC

F970 01FFFF LXI B, -1
F973 D5 PUSH D
F974 7E MOV A, M
F975 A9 XRA C
F976 57 MOV D, A
F977 0F RRC
F978 0F RRC
F979 0F RRC
F97A 0F RRC
F97B E60F ANI 0FH
F97D AA XRA D
F97E 5F MOV E, A
F97F 0F RRC
F980 0F RRC
F981 0F RRC
F982 57 MOV D, A
F983 E61F ANI 1FH
F985 A8 XRA B
F986 4F MOV C, A
F987 7A MOV A, D
F988 E6E0 ANI 0EOH
F98A A8 XRA E
F98B 47 MOV B, A
F98C 7A MOV A, D
F98D 0F RRC
F98E E6F0 ANI 0F0H
F990 A9 XRA C
F991 4F MOV C, A
F992 23 INX H
F993 D1 POP D
F994 7A MOV A, D
F995 BC CMP H
F996 D8 RC
F997 C273F9 JNZ CREECH+3
F99A 7B MOV A, E
F99B BD CMP L
F99C D8 RC
F99D C373F9 JMP CREECH+3

;
SOFTWARE UART ROUTINES

SERIALIN:

MVI B, 1 ; SET TO SUPPRESS OUTPUT IN DF
LHLD SPEED ; GET SPEED CONSTANT
PUSH H ; SAVE ON STACK
MVI E, 0FFH ; INITIALIZE 1/2 THE SPEED
LDA DISKFUNCTION ; LOOK FOR SPEED
RAR ; ROTATE INTO CARRY
RAR
RAR JNC SLOOK ; IS SERIAL INPUT BIT
CALL SERIALDELAY ; WAIT HALF A
POP H ; RESET SPEED CONSTANT
LDA DISKFUNCTION ; VERIFY THAT
RAR ; IS STILL PRESENT
RAR
RAR JNC SWAIT
MVI D, 0FFH ; INITIALIZE OTHER HA

GTBIT:

PUSH H ; UPDATE THE STACK
DAD H ; CALCULATE THE SPEED
DCX H ; CONSTANT FOR A FULL
LDA DISKFUNCTION ; GET THE
RAR ; ROTATE TO BIT ZERO
MOV E, A ; UPDATE THE SHIFT REGISTER
CALL SERIALDELAY ; DELAY ONCE
POP H ; GET THE SPEED CONSTANT
JC GTBIT ; HAS THE START BIT SET
MOV A, D ; MOVE BYTE TO ACC
ANI 7FH ; CLEAR HIGH BIT
RET

SERIALOUT:

MOV A, C ; MOVE CHARACTER TO ACC
ADD A ; ADD A START BIT
MOV B, A ; MAKE BIT 0 OF B A ZERO
MOV E, A ; SHIFTED DATA TO E
MVI A, 11 ; THIS IS THE BIT COUNT
MOV C, A ; COUNT TO REG C
RAL ; LOAD D WITH THE REST
MOV D, A ; BITS AND HIGH ORDER

OLOOP:

LHLD SPEED ; GET THE SPEED CONSTANT
LHLD SPEED ; PADDING
DAD H ; ADJUST FOR OUTPUT
DCX H ; LOOP
CALL SERIALDELAY ; OUTPUT DATA
DCR C ; DECREMENT BIT COUNT
JNZ OLOOP
RET
SERIALDELAY:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F9EA</td>
<td>7B</td>
<td>MOV A,E</td>
</tr>
<tr>
<td>F9EB</td>
<td>B0</td>
<td>ORA B</td>
</tr>
<tr>
<td>F9EC</td>
<td>0F</td>
<td>RRC B</td>
</tr>
<tr>
<td>F9ED</td>
<td>0F</td>
<td>RRC B</td>
</tr>
<tr>
<td>F9EE</td>
<td>0F</td>
<td>RRC B</td>
</tr>
<tr>
<td>F9EF</td>
<td>3203FB</td>
<td>STA SERIALOUTPUT</td>
</tr>
<tr>
<td>F9F2</td>
<td>2D</td>
<td>DCR L</td>
</tr>
<tr>
<td>F9F3</td>
<td>00</td>
<td>NOP</td>
</tr>
<tr>
<td>F9F4</td>
<td>C2EAF9</td>
<td>JNZ SERIALDELAY</td>
</tr>
<tr>
<td>F9F7</td>
<td>7A</td>
<td>MOV A,D</td>
</tr>
<tr>
<td>F9F8</td>
<td>1F</td>
<td>RAR</td>
</tr>
<tr>
<td>F9F9</td>
<td>7B</td>
<td>MOV A,E</td>
</tr>
<tr>
<td>F9FA</td>
<td>1F</td>
<td>RAR</td>
</tr>
<tr>
<td>F9FB</td>
<td>5F</td>
<td>MOV E,A</td>
</tr>
<tr>
<td>F9FC</td>
<td>7A</td>
<td>MOV A,D</td>
</tr>
<tr>
<td>F9FD</td>
<td>1F</td>
<td>RAR</td>
</tr>
<tr>
<td>F9FE</td>
<td>57</td>
<td>MOV D,A</td>
</tr>
<tr>
<td>F9FF</td>
<td>C9</td>
<td>RET</td>
</tr>
</tbody>
</table>
A-TYPE MEMCBIOS

B-TYPE MEMCBIOS.PRN

; CBOS FOR MICROMATION 16K VERSION OF CP/M VERSION 1.3
; COPYRIGHT (C) 1977, MICROMATION AND DIGITAL RESEARCH
; FEB 17, 1978
; DRIVERS FOR MEMOREX DRIVE

0010 = MSIZE EQU 16 ; SIZE OF OPERATING SYSTEM IN K
; (CURRENTLY 16K). THIS NUMBER
; CHANGED FOR LARGER SYSTEMS.
3E00 = LOCATION EQU MSIZE*1024-512 ; ORG LOCATION FOR THE
; ORG LOCATION ; BASE OF BIOS IN 16K SYSTEM

; MAP OF SCRATCH AREA
FA00 = SCRATCH EQU 0FA00H ; BASE ADDR OF RAM SCRATCH
FA10 = PRESDSK EQU SCRATCH+10H
FA6F = BOOTSTACK EQU SCRATCH+6FH
FA70 = CONTROLBYTE EQU SCRATCH+70H
FA73 = RETRYCOUNT EQU SCRATCH+73H
FA74 = DMAADDR EQU SCRATCH+74H
FA76 = ADDRPRTR EQU SCRATCH+76H
FA77 = TRACK EQU SCRATCH+77H
FA79 = SECTOR EQU SCRATCH+79H

; PSEUDO PORTS IN ROM
FB00 = DATAPORT EQU SCRATCH+100H
FB01 = MARKPORT EQU SCRATCH+101H
FB02 = DISKFUNCTION EQU SCRATCH+102H
FB03 = LOADPORT EQU SCRATCH+103H
FB03 = SERIALOUTPORT EQU SCRATCH+103H

F822 = HOME EQU 0F822H
F88A = SELDSK EQU 0F88AH
F845 = SETTRK EQU 0F845H
F956 = SETSEC EQU 0F956H ; SET SECTOR NUMBER
F964 = SETDMA EQU 0F964H

F8E3 = DISKWRITE EQU 0F8E3H
F8C1 = DISKREAD EQU 0F8C1H
F8D2 = DATAFPER EQU 0F8D2H

FA71 = SPEED EQU SCRATCH+71H
F9D1 = SERIALOUT EQU 0F9D1H
F9A0 = SERIALIN EQU 0F9A0H
; CBASE EQU (MSIZE-16)*1024 ; BIAS FOR SYSTEMS GREATER THAN 
; CPMB EQU CBASE+2900H 
; BDOE EQU CBASE+3106H 
; CPMB-128 
; CPML EQU $-CPMB 
; NSECTS EQU 2AH ; CHANGE FOR LARGER MEMORY 

; C32D3E JMP COLDBOOT 
; C3553E JMP WBOOT 
; C3023F JMP CONST 
; C31B3F JMP CONIN 
; C32F3F JMP CONOUT 
; C3333F JMP LIST 
; C3333F JMP PUNCH 
; C3333F JMP READER 
; C332F8 JMP HOME 
; C3343E JMP IOSELDSK 
; C345F8 JMP SETTRK 
; C356F9 JMP SETSEC 
; C364F9 JMP SETDMA 
; C3343F JMP READ 
; C3503F JMP WRITE 

; C32D3E XRA A 
; C3210FA STA PRESDK; INITIALIZE PRESENT DISK 
; C3DA3E JMP BOOT 

; C3100 XA H,SCRATCH ; POINT TO BOTTOM OF SCRATCH 
; C3A10FA LDA PRESDK ; GET PRESENT DRIVE # 
; C3F10 CPI 10H ; CHECK FOR VALID # 
; C32B3E JNC GOSELDSDK ; GET OUT IF INVALID 
; C36F L,A MOV L,A ; POINT TO TRACK OF PRESENT DRIVE 
; C32C INR L ; INCREMENT TO NEXT BYTE 
; C377FA LDA TRACK ; GET PRESENT TRACK 
; C377 MOV M,A ; STORE IN BUFFER 
; C369 MOV L,C ; POINT TO SELECTED DRIVE BUFFER 
; C32C INR L ; NEXT BYTE 
; C37E MOV A,M ; GET TRACK OF SELECTED DRIVE 
; C327FA STA TRACK ; UPDATE CONTROLLER 

; C327FA MOV A,C ; LOAD SELECTED DRIVE 
; C3210FA STA PRESDK ; UPDATE DISK BUFFER 
; C338AF8 JMP SELDSK ; GO TO CONTROLLER 

; C9 ERRORV: RET ; NOT CURRENTLY USED 
; 00 NOP ; RESERVED FOR FUTURE ERROR REPORTING 
; 00 NOP 

; 318000 WBOOT: LXI SP,80H ; GET PRESENTLY SELECTED DRIVE 
; 3A10FA LDA PRESDK ; STORE IN BUFFER 

; 32D93E STA CURRDRIVE ; SELECT DRIVE A TO REBOOT 

; 0E00 MVI C,0 
; CD8AF8 CALL SELDSK 
; CD22F8 CALL HOME
READ DISKETTE FOR TWO TRACKS, STARTING AT ROOT LOAD:

3E66 018028
3E69 CD64F9
3E6C 110000
3E6F 7B
3E70 FE1A
3E72 CAAD3E

RDTRK: ;READ THE FIRST/NEXT TRACK
3E75 21BF3E
3E78 19
3E79 7E
3E7A 13
3E7B D5
3E7C 4F
3E7E F5
3E7F CD56F9
3E82 F1
3E83 E1
3E84 E5
3E85 118000

GET SKewed SECTOR NUMBER
3E88 3D
3E89 CA903E
3E8C 19
3E8D C3883E

MUL: ;HL IS DMA ADDRESS FOR THIS SECTOR
3E90 2274FA
3E93 3A77FA
3E96 B7
3E97 CAA23E
3E9A 3A79FA
3E9D D612
3E9F F2A83E

RELP:
3EA2 CD343F
3EA5 C25E3E

SKIPREAD:
3EA8 C1
3EA9 D1
3EAA C36F3E

NXTTRK:
3EAD 3A77FA
3EB0 B7
3EB1 C2DA3E
3EB4 0E01
3EB6 CD45F8
3EB9 018035
3EBC C36C3E

PLAY: ;TO READ THE ENTIRE TRACK
TRAN: TRANSLATION TABLE FOR SKEW FACTOR

<table>
<thead>
<tr>
<th>Code</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>3EBF</td>
<td>01H</td>
</tr>
<tr>
<td>3EC0</td>
<td>05H</td>
</tr>
<tr>
<td>3EC1</td>
<td>09H</td>
</tr>
<tr>
<td>3EC2</td>
<td>0DH</td>
</tr>
<tr>
<td>3EC3</td>
<td>11H</td>
</tr>
<tr>
<td>3EC4</td>
<td>15H</td>
</tr>
<tr>
<td>3EC5</td>
<td>19H</td>
</tr>
<tr>
<td>3EC6</td>
<td>03H</td>
</tr>
<tr>
<td>3EC7</td>
<td>07H</td>
</tr>
<tr>
<td>3EC8</td>
<td>0BH</td>
</tr>
<tr>
<td>3EC9</td>
<td>0FH</td>
</tr>
<tr>
<td>3ECA</td>
<td>13H</td>
</tr>
<tr>
<td>3ECB</td>
<td>17H</td>
</tr>
<tr>
<td>3ECC</td>
<td>02H</td>
</tr>
<tr>
<td>3ECD</td>
<td>06H</td>
</tr>
<tr>
<td>3ECF</td>
<td>10H</td>
</tr>
<tr>
<td>3ED0</td>
<td>0AH</td>
</tr>
<tr>
<td>3ED1</td>
<td>0EH</td>
</tr>
<tr>
<td>3ED2</td>
<td>12H</td>
</tr>
<tr>
<td>3ED3</td>
<td>16H</td>
</tr>
<tr>
<td>3ED4</td>
<td>0AH</td>
</tr>
<tr>
<td>3ED5</td>
<td>0EH</td>
</tr>
<tr>
<td>3ED6</td>
<td>12H</td>
</tr>
<tr>
<td>3ED7</td>
<td>16H</td>
</tr>
<tr>
<td>3ED8</td>
<td>0AH</td>
</tr>
<tr>
<td>3ED9</td>
<td>0EH</td>
</tr>
<tr>
<td>3EDC</td>
<td>12H</td>
</tr>
<tr>
<td>3EDD</td>
<td>16H</td>
</tr>
<tr>
<td>3EED</td>
<td>0AH</td>
</tr>
<tr>
<td>3EE0</td>
<td>0BH</td>
</tr>
<tr>
<td>3EE1</td>
<td>0CH</td>
</tr>
<tr>
<td>3EE2</td>
<td>0DH</td>
</tr>
<tr>
<td>3EE3</td>
<td>11H</td>
</tr>
<tr>
<td>3EE4</td>
<td>15H</td>
</tr>
<tr>
<td>3EE5</td>
<td>0FH</td>
</tr>
<tr>
<td>3EE6</td>
<td>13H</td>
</tr>
<tr>
<td>3EE7</td>
<td>17H</td>
</tr>
<tr>
<td>3EE8</td>
<td>02H</td>
</tr>
<tr>
<td>3EE9</td>
<td>06H</td>
</tr>
<tr>
<td>3EEA</td>
<td>10H</td>
</tr>
<tr>
<td>3EEB</td>
<td>14H</td>
</tr>
<tr>
<td>3EEC</td>
<td>18H</td>
</tr>
</tbody>
</table>

; CURRDRIVE DB 0

; SET THE SOFTWARE UART SPEED
; IF THE CBIOS IS MODIFIED FOR AN I/O BOARD THE
; CODE TO PROGRAM THE UART SHOULD BE
; PUT HERE AND THE INSTRUCTION TO SET THE SOFTWARE
; UART SPEED REMOVED.

3EDA 210700 LXI H,0007H
3EDD 2271FA SHLD SPEED

3EE0 3EC3 MVI A,0C3H
3EE2 320000 STA 0
3EE5 21033E LXI H,EBOOT
3EE8 220100 SHLD 1
3EEB 320500 STA 5
3EEE 210631 LXI H,BDOS
3EF1 220600 SHLD 6
3EF4 018000 LXI B,80H
3EF7 CD64F9 CALL SETDMA
3EFA FB FI.
3EFB 3AD93E LDA CURRDRIVE ;ACTIVE DISK
3EE4 4F MOV C,A
3EFF C30029 JMP CPMB

;SOFTWARE UART CONSOLE ROUTINES

CONST:

3F02 060A MVI B,10 ;EACH LOOP = 35 MICROSECONDS

CONST1:

3F04 3A02FB LDA DISKFUNCTION ;LOOK FOR BIT
3F07 1F RAR
3F08 1F RAR
3F09 D2123F JNC CONSTFD ;FOUND BIT
3F0C 05 DCR B
3F0D C2043F JNZ CONST1 ;LOOP FOR SPECIFIED TIME
3F10 AF XRA A ;ZERO A - NO CHARACTER FOUND
ERROR CHECKING READ AND WRITE RTNS FOR
MICROMATION CBIOs

AUGUST 24, 1977

0014 = RETRYLIMIT EQU 20 ;NUMBER OF RETRIES

READ:
3F34 CDA83F CALL READYNOW ;CALL PROM RTN
3F37 AF XRA A ;GET A ZERO
3F38 3273FA STA RETRYCOUNT

RETRYREAD:
3F3B CDC1F8 CALL DISKREAD ;CALL PROM RTN
3F3E 3E00 MVI A,0 ;ZERO ACCUM, LEAVE PSW
3F40 C8 RZ
3F41 CD523E CALL ERRORV
3F44 CD7A3F CALL ERRORCHECK
3F47 C23B3F JNZ RETRYREAD ;IF ERROR RETRY
3F4A CDD2F8 CALL DATAXFER ;TRANSFERS DATE
3F4D 3E0F MVI A,0PH ;ERROR CODE
3F4F C9 RET ;EITHER RETRY SUCCESS

WRITE:
3F50 CDA83F CALL READYNOW ;READ STATUS
3F53 3A02FB LDA DISKFUNCTION
3F56 E604 ANI 04H ;CHECK WRITE PROTECT
3F58 CA643F JZ NOTPROTECT
3F5B 11D33F LXI D,WPMSG
3F5E CDBB3F CALL PRINTMSG
3F61 C30000 JMP 0 ;WARM BOOT

NOTPROTECT:
3F64 AF XRA A ;GET A ZERO
3F65 3273FA STA RETRYCOUNT

RETRYWRITE:
3F68 CDE3F8 CALL DISKWRITE ;CALL PROM RTN
3F6B 3E00 MVI A,0 ;ZERO ACCUM, LEAVE PSW
3F6D C8 RZ ;IF NO ERROR THEN RP
CALL ERRORV  
CALL ERRORCHECK  ; IF ERROR, RETRY  
JNZ RETRYWRITE  ; RETURN ERROR CODE  
MVI A, 0FH  
RET  
ERRORCHECK:  
LDA RETRYCOUNT  ; GET NUMBER OF RETRY  
INR A  ; ADD ONE  
STA RETRYCOUNT  ; HAVE WE RETRIED ENOUGH?  
CPI RETRYLIMIT+1  ; IF YES, RETURN WRITE PROTECTED  
RZ  ; IS ERROR A TRACK ERROR?  
MVI A, 77H  ; L HOLDS LOCATION OF ERROR TRACK  
CMP L  ; IF NOT AT 77 THEN  
NZ  
; TRACK ERROR  
LDA RETRYCOUNT  ; GET NUMBER OF RETRY  
SUI 10  
; ADD ONE  
RM  ; SAVE TRACK AND SECTOR  
LDA TRACK  ; GET TRACK IN C  
MOV C, A  ; GET TRACK IN C  
LDA SECTOR  ; GET SECTOR IN B  
MOV B, A  ; GET SECTOR IN B  
PUSH B  ; SAVE TRACK AND SECTOR  
CALL HOME  ; PROM RTN TO HOME  
POP B  ; RESTORE TRACK AND SECTOR  
PUSH B  ; SAVE TRACK AND SECTOR  
CALL SETrk  ; PROM RTN TO FIND TRACK  
POP B  ; GET TRACK AND SECTOR  
MOV C, B  ; GET SECTOR IN REG (C)  
CALL SETSEC  ; PROM RTN TO FIND SECTOR  
MVI A, 0FH  ; TURN OFF THE ZERO FLAG FOR TRACK ERROR  
DCR A  
RET  
READYNOW:  
LDA DISKFUNCTION  ; CHECK STATUS  
RLC  ; CHECK READY LINE  
RC  ; CARRY SET, DRIVE READY  
LXI D, NOTRDYM  
CALL PRINTMSG  ; POINT TO MSG BUFF  
READYLOOP:  
LDA DISKFUNCTION  
RLC  
RC  
JMP READYLOOP  ; LOOP TILL READY  
PRINTMSG:  
LDAX D  ; GET FIRST CHARACTER  
CPI 'S'  ; END DELIMITER?  
RZ  ; RET IF DONE  
PUSH D  
MOV C, A  
CALL CONOUT  
POP D  
INX D  
JMP PRINTMSG  ; LOOP UNTIL DONE  
NOTRDYM:  
JMP PRINTMSG  ; LOOP UNTIL DONE  
WPMSG:  
DB 'NOT READY$'  
WPM:  
DB 'WRITE PROTECTED$'
If the CBIOS is not changed, the following procedure may be used to generate new systems for any size memory.

The file CPM.COM has been included with this diskette, which enables you to generate a CP/M system for any memory size, up to 64K bytes. The command

CPM <cr>

(where <cr> denotes the carriage-return key) loads the CPM.COM program and gives it control. This program then examines the current memory configuration, and produces a new CP/M system which is relocated to the top of the memory (actually, the highest contiguous RAM area is used). The newly constructed CP/M system then gets control, and the system starts with the normal sign-on message.

The command

CPM **

constructs a new version of the CP/M system, but leaves it in memory, ready for a sysgen operation. The message

READY FOR "SYSGEN" OR "SAVE 32 CPMxx.COM"

is printed at the console upon completion, where xx is the memory size in kilobytes. The operator can then type

SYSGEN

with the response

GET SYSTEM (Y/N)?n user must respond with "n"

and the message

PUT SYSTEM (Y/N)?y user must respond with "y"

DESTINATION ON B, THEN TYPE RETURN

Place the new diskette on drive B, and type a return when ready (note that if you answer with an "a" rather than a "y" to the prompt above, SYSGEN will place the CP/M system on drive A instead of drive B). Syssgen will then type

FUNCTION COMPLETE, REBOOTING

The user can then go through the reboot process with the old or new diskette.

The operator could also have typed

SAVE 32 CPMxx.COM

at the completion of CPM.COM, which would place the CP/M memory image on disk. In this case, the relocated memory image can be "patched" to include custom I/O drivers, as described in the CP/M Alteration Guide.

(over)
Note that the memory size can be given explicitly to the CPM.COM program when it is started in order to override the internal mechanisms which determine the amount of memory on the system. In this case, the operator must type

```
CPM xx
```

where xx is the memory size in decimal kilobytes. The first form produces a CP/M system which operates in xx kilobytes, and starts the newly created system when the relocation is complete. The second form creates the new system, but leaves it in memory for a sysgen or save operation.

For example, the invocation

```
CPM 48 *
```

starts CPM.COM, and creates a 48K system in memory. Upon completion, the message

```
READY FOR "SYSGEN" OR
"SAVE 32.CPM48.COM"
```

is typed. The operator can then perform the sysgen or save operation as described above. Note that the newly created system is serialized with the diskette attached to your original diskette, and are subject to the conditions of the Software Licensing Agreement included in this package.