MICROPROCESSOR DEVELOPMENT SYSTEM FOR THE ALTOS SERIES MICROCOMPUTER

JUN 81 S H HUGHES
THESIS

A Microprocessor Development System for the ALTOS Series Microcomputers

by

Stephen Michael Hughes

June 1981

Thesis Advisor: M. L. Cotton

Approved for public release; distribution unlimited
**Title**: A Microprocessor Development System for the ALTOS Series Microcomputers.

**Author**: Stephen Michael Hughes

**Performing Organization**: Naval Postgraduate School, Monterey, California 93940

**Report Date**: June 1981

**Abstract**: An ALTOS series microcomputer is being used as the host computer in a microprocessor development system (MDS). The MDS hardware, consisting of the PRO-LOG STD bus, a Z80 CPU card, 2K bytes EPROM and 36K bytes random access memory, is controlled by the host via a single serial I/O port. The system provides the capability to develop and test both software and hardware in the combined CP/M (MP/M) and MDS environments.
A Microprocessor Development System for the ALTOS Series Microcomputers

by

Stephen Michael Hughes
Lieutenant, United States Navy
B.S., United States Naval Academy, 1975

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1981

Author: Stephen Michael Hughes

Approved by: Mitchell A. Cotton
Thesis Advisor

R. D. Stone
Second Reader

R. D. Stone
Acting Chairman, Department of Electrical Engineering

William M. Tolles
Dean of Science and Engineering

2
ABSTRACT

An ALTOS series microcomputer is being used as the host computer in a microprocessor development system (MDS). The MDS hardware, consisting of the PRO-LOG STD bus, a Z80 cpu card, 2K bytes EPROM and 36K bytes random access memory, is controlled by the host via a single serial I/O port. The system provides the capability to develop and test both software and hardware in the combined CP/M (MP/M) and MDS environments.
# TABLE OF CONTENTS

I. INTRODUCTION .......................................................... 6

II. THE MICROPROCESSOR DEVELOPMENT SYSTEM ...................... 8
   A. HARDWARE CONSIDERATIONS ........................................ 9
   B. SOFTWARE CONSIDERATIONS ......................................... 14
   C. THE SYSTEM CONTROL SOFTWARE .................................. 14
       1. The HOST Control Software .............................. 15
       2. The MLS Onboard Monitor ................................. 22

III. SYSTEM IMPLEMENTATION AND CUSTOMIZATION .................... 24
   A. PUTTING IT ALL TOGETHER .................................... 24
   B. CUSTOMIZATION .................................................. 27
   C. SYSTEM LIMITATIONS .......................................... 31

IV. CONCLUSIONS AND RECOMMENDATIONS .............................. 36
   A. FUTURE HARDWARE ............................................. 36
   B. FUTURE SOFTWARE ............................................... 37

APPENDIX A: AMES USERS GUIDE ...................................... 39
APPENDIX B: HOST AND MLS FLOW CHARTS FOR USER OPTIONS ........ 61
APPENDIX C: AMES HOST CONTROL SOFTWARE LISTING ............... 71
APPENDIX D: MLS MONITOR SOFTWARE LISTING ......................... 120
APPENDIX E: MDS MEMORY TEST PROGRAM LISTING .................... 129
APPENDIX F: SAMPLE MENU LISTING .................................. 144
APPENDIX G: SAMPLE BASIC INSTRUCTION LISTING .................... 145
APPENDIX H: SAMPLE INPUT PARAMETER FORMAT LISTING ............ 146

BIBLIOGRAPHY .................................................................. 148

INITIAL DISTRIBUTION LIST ............................................ 149
## List of Figures

1. **PRO-LOG STD BUS PIN DEFINITIONS** .......................... 12
2. **HOST CONTROL PROGRAM** .................................. 18
3. **RS-232C PIN DEFINITIONS AND SYSTEM I/O SETUP** .... 26
4. **INTEL HEX FILE RECORD FORMAT** .......................... 33
I. INTRODUCTION

The Naval Postgraduate School Electrical Engineering Department's microcomputer/microprocessor development laboratory, presently being used for microprocessor application courses at the beginning and intermediate levels, offers two methods of applications development. One method uses the Tektronix 2042 development system. While this system is very capable for hardware applications development, it is limited in available software, provides for use by only a single user at a time, and takes a considerable amount of time to learn to use properly. Also, because of the high cost of additional in-circuit emulation modules for different processors, the system has been slow to expand. On the other end of the spectrum, the ALTOS series single and multi-user microcomputer systems provide extremely good support for software development due to the vast variety of CP/M based software currently available. These systems have a much lower per-user cost and provide a work environment more enhancing to individual productiveness. The primary disadvantage, however, is the lack of support for hardware development, without having to get inside the computers and building some type of kludged interface whose reliability is often haphazard at best.
The design and implementation of a relatively low cost, low complexity, highly flexible microprocessor development system, combining many of the good features of each of these methods is the topic of further discussion in this thesis.
II. THE MICROPROCESSOR DEVELOPMENT SYSTEM

The bounding needs of this microprocessor development system (MDS) are grouped into the four areas listed below:

The overall system cost should be relatively low in contrast to large development systems such as the Tektronix 3002.

The MDS should be of low complexity in both software and hardware requirements.

The system should utilize existing software and hardware to the best extent possible.

The system should be expandable and easily customized or reconfigured to operate with numerous other microcomputer systems.

The determination of these needs made the selection of final requirements almost automatic. The primary decisions were what capabilities should be included in the MDS within the constraints of the needs given and the time available. Typical development system components include software support for editing, assembling and debugging applications programs and hardware support for testing both the software and hardware in an in-circuit emulation (ICE) environment.

Because of the low complexity constraint and the limited time available for this project, it was decided that the ICE component would be the area where most of the compromises would be made during the system design. To further meet the
stated needs, the decision was made to design the system for operation as a task in the CP/M and MP/M operating systems environment.

A. HARDWARE CONSIDERATIONS

Initial ideas for meeting the hardware needs of the MDS included utilizing an ALTOS microcomputer as the control computer for a separate hardware development system. The minimum hardware development system would consist of a dedicated microprocessor, EPROMs for an onboard monitor, sufficient random access memory (RAM) for storage and execution of fairly complex programs and a serial RS-232C port for interface to the ALTOS.

The ALTOS computer and the hardware development system together would form the complete microprocessor development system. For clarity, the ALTOS computer will henceforth be referred to as the 'HOST', the hardware development system as the 'MDS' and the overall system as the 'AMDS', for ALTOS Microprocessor Development System.

The MDS hardware was the subject of primary consideration during the initial stages of system design. Consideration was first given to wire-wrapping circuits to meet the stated minimum hardware requirements, but this approach was soon recognized as being prohibitive due to the considerable time requirements involved for this type of work.
This approach would also contribute to a less reliable and less flexible system for long term use and future expansion.

Thus, the decision was made to use a standardized bus system which has achieved industry acceptance in both proven applications and in manufacturer support and which would offer a reasonable initial system cost (under $1500.00). While several manufacturers offer such a system, the PRO-LOG Corporation STD bus was chosen over others primarily due to its immediate availability and local manufacturer support.

The final MDS hardware configuration consists of the following PRO-LOG components:

- A 16 slot STD bus and card cage with provisions for wire-wrapped cards.
- A 2MHz Z80 processor card with onboard provisions for up to 4K bytes of RAM and up to 8K bytes of 2716 EPROM.
- Two 16K byte static memory cards.
- A dual USART card consisting of two fully independent, asynchronous RS-232C serial ports with provision for one of these to be configured as a 20mA loop for TTY applications.
- Several blank utility cards for wire-wrapped applications.
- A DC power supply providing +5V/10A and ±12V/1A.

The only hardware modification necessary to get this system operable was the addition of a manual reset switch which is only a momentary ground to the push-button reset.
pin (48) on the STD bus. The STD bus pin definitions are given in Figure 1.
<table>
<thead>
<tr>
<th>PIN</th>
<th>MNEMONIC</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+5VDC</td>
<td>Logic Power</td>
</tr>
<tr>
<td>2</td>
<td>+5VDC</td>
<td>Logic Power</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
<td>Logic Ground</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
<td>Logic Ground</td>
</tr>
<tr>
<td>5</td>
<td>VBB#1</td>
<td>Logic Bias #1 (-5V)</td>
</tr>
<tr>
<td>6</td>
<td>VBB#2</td>
<td>Logic Bias #2 (-5V)</td>
</tr>
<tr>
<td>7</td>
<td>D3</td>
<td>Data Bit 3</td>
</tr>
<tr>
<td>8</td>
<td>D7</td>
<td>Data Bit 7</td>
</tr>
<tr>
<td>9</td>
<td>D2</td>
<td>Data Bit 2</td>
</tr>
<tr>
<td>10</td>
<td>D6</td>
<td>Data Bit 6</td>
</tr>
<tr>
<td>11</td>
<td>D1</td>
<td>Data Bit 1</td>
</tr>
<tr>
<td>12</td>
<td>D5</td>
<td>Data Bit 5</td>
</tr>
<tr>
<td>13</td>
<td>D0</td>
<td>Data Bit 0</td>
</tr>
<tr>
<td>14</td>
<td>D4</td>
<td>Data Bit 4</td>
</tr>
<tr>
<td>15</td>
<td>A7</td>
<td>Address Line 7</td>
</tr>
<tr>
<td>16</td>
<td>A15</td>
<td>Address Line 15</td>
</tr>
<tr>
<td>17</td>
<td>A6</td>
<td>Address Line 6</td>
</tr>
<tr>
<td>18</td>
<td>A14</td>
<td>Address Line 14</td>
</tr>
<tr>
<td>19</td>
<td>A5</td>
<td>Address Line 5</td>
</tr>
<tr>
<td>20</td>
<td>A13</td>
<td>Address Line 13</td>
</tr>
<tr>
<td>21</td>
<td>A4</td>
<td>Address Line 4</td>
</tr>
<tr>
<td>22</td>
<td>A12</td>
<td>Address Line 12</td>
</tr>
<tr>
<td>23</td>
<td>A3</td>
<td>Address Line 3</td>
</tr>
<tr>
<td>24</td>
<td>A11</td>
<td>Address Line 11</td>
</tr>
<tr>
<td>25</td>
<td>A2</td>
<td>Address Line 2</td>
</tr>
<tr>
<td>26</td>
<td>A10</td>
<td>Address Line 10</td>
</tr>
<tr>
<td>27</td>
<td>A1</td>
<td>Address Line 1</td>
</tr>
<tr>
<td>28</td>
<td>A9</td>
<td>Address Line 9</td>
</tr>
<tr>
<td>29</td>
<td>A0</td>
<td>Address Line 0</td>
</tr>
<tr>
<td>30</td>
<td>A8</td>
<td>Address Line 8</td>
</tr>
<tr>
<td>31</td>
<td>WR*</td>
<td>Write to Memory or I/O</td>
</tr>
<tr>
<td>32</td>
<td>RD*</td>
<td>Read Memory or I/O</td>
</tr>
<tr>
<td>33</td>
<td>IORQ*</td>
<td>I/O Address Select</td>
</tr>
<tr>
<td>34</td>
<td>MEMRO*</td>
<td>Memory Address Select</td>
</tr>
<tr>
<td>35</td>
<td>IOEXP</td>
<td>I/O Expansion</td>
</tr>
<tr>
<td>36</td>
<td>MEMEX</td>
<td>Memory Expansion</td>
</tr>
<tr>
<td>37</td>
<td>REFRESH*</td>
<td>Refresh Timing</td>
</tr>
<tr>
<td>38</td>
<td>MCSYNC*</td>
<td>CPU Machine Cycle Sync.</td>
</tr>
<tr>
<td>39</td>
<td>STATUS 1*</td>
<td>CPU Status</td>
</tr>
<tr>
<td>40</td>
<td>STATUS 0*</td>
<td>CPU Status</td>
</tr>
</tbody>
</table>

Figure 1 - PRO-LOG STD Bus Pin Definitions
<table>
<thead>
<tr>
<th>PIN</th>
<th>MNEMONIC</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>BUSAK*</td>
<td>Bus Acknowledge</td>
</tr>
<tr>
<td>42</td>
<td>BUSRQ*</td>
<td>Bus Request</td>
</tr>
<tr>
<td>43</td>
<td>INTAK*</td>
<td>Interrupt Acknowledge</td>
</tr>
<tr>
<td>44</td>
<td>INTRQ*</td>
<td>Interrupt Request</td>
</tr>
<tr>
<td>45</td>
<td>WAITRQ*</td>
<td>Wait Request</td>
</tr>
<tr>
<td>46</td>
<td>NMIRQ*</td>
<td>Nonmaskable Interrupt</td>
</tr>
<tr>
<td>47</td>
<td>SYSRESET*</td>
<td>System Reset</td>
</tr>
<tr>
<td>48</td>
<td>PBRESET*</td>
<td>Push-Button Reset</td>
</tr>
<tr>
<td>49</td>
<td>CLOCK*</td>
<td>Clock from Processor</td>
</tr>
<tr>
<td>50</td>
<td>CNTRL*</td>
<td>AUX Timing</td>
</tr>
<tr>
<td>51</td>
<td>PCO</td>
<td>Priority Chain Out</td>
</tr>
<tr>
<td>52</td>
<td>PCI</td>
<td>Priority Chain In</td>
</tr>
<tr>
<td>53</td>
<td>AUX GND</td>
<td>AUX Ground</td>
</tr>
<tr>
<td>54</td>
<td>AUX GND</td>
<td>AUX Ground</td>
</tr>
<tr>
<td>55</td>
<td>AUX +V</td>
<td>AUX Positive (+12VDC)</td>
</tr>
<tr>
<td>56</td>
<td>AUX -V</td>
<td>AUX Negative (-12VDC)</td>
</tr>
</tbody>
</table>

*Low-level active indicator*
B. SOFTWARE CONSIDERATIONS

The editing, assembling and debugging software needs for the AMDS were easily fulfills by deciding to utilize CP/M based software. The basic CP/M and MP/M operating systems provide software for each of these needs, therefore simplifying the overall system design considerably. Additionally, the existence of a vast selection of CP/M based software products on the commercial market greatly enhances the growth prospects for software applications development with this system. An added feature of the decision to use CP/M based software is the ability to develop and test software on any microcomputer using the CP/M operating system. This feature alone is one of the most advantageous aspects of the AMDS.

With these capabilities accounted for, the remaining software considerations were those of determining the software requirements for the HOST to control the MDS and deciding upon those capabilities which should be included in the control software package.

C. THE SYSTEM CONTROL SOFTWARE

The system control software needs were divided into two areas: 1) the control program resident in the HOST, to be used in exercising overall control of both the ALTOS and the MDS and; 2) the MDS onboard monitor program, to be used for communications with the HOST and for interpreting and executing HOST commands.
1. The HOST Control Software

The primary functions of the AMDS control program resident in the HOST are to communicate with the system user and to exercise positive control of the MIS. It is intended to be the workhorse of the system, providing numerous routines to simplify the work required of the MIS.

A study of the monitor and control programs for typical development systems helped in identifying the following software needs as the most essential user requirements for implementation into the HOST control program:

A routine to download data from disk to MLS memory.
A routine to upload data from MDS memory and store it on disk.
A routine for examining and modifying MDS memory contents.
A routine for filling specified blocks of MDS memory with a specific byte of data for memory initialization.
A routine to locate a specific data sequence in MDS memory.
A routine to dump the contents of MIS memory to a CRT or printer in a format conducive to user interpretation.
A routine to initiate the execution of a program previously placed into MIS memory.

Each of these routines are implemented in the HOST control program. Additional routines provide: 1) the ability to perform additions and subtractions of two hexadecimal
numbers and display the results, 2) a routine for continuous modification of MES memory without an intermediate examination of each location, and 3) routines for online user self-help and system use instructions.

The primary consideration in the design of the HOST control program was in making it user oriented. Thus, considerable effort was made to make the system easy to learn and to provide positive user feedback in all modes of operation. Examples of this include the implementation of a menu displaying all user options, detailed instructions for required input formats (available at any time), and fully explanatory error displays. Operation of the system is designed so that the user should never be in doubt as to what is going or what is required of him.

The control program flow is straightforward. Program parameters are first initialized followed by displaying the menu of options on the user's console and prompting him for input of the desired option. The input is then interpreted and a branch is made to the routine chosen, whereupon the user is again prompted for additional input unique to that option. Upon completion of the option, at the command of the user or after a trap to certain errors, the program returns control to the menu routine to await further user commands. This flow is easier visualized, as shown in Figure 2.

The flow of the individual option subroutines is equally simple. Upon entering each routine, again various
parameters are initialized and the user is prompted for initial input. When the proper input is received, the routine takes the necessary actions to perform the task, including communications with the MDS, if applicable, and prompting the user for additional inputs as required. On completion of the option, control returns to the menu routine.
Figure 2 - HOST Control Program Flowchart
All user input is checked for validity including proper syntax, correct number and placement of parameter delimiters and for valid hexadecimal digits where applicable. Additionally, the input is checked for user requests for help or to terminate the option and return to the menu. Data input and output formats were kept as compatible as possible with those in the CP/M dynamic debugging tool (LLT). All input is terminated with a carriage return or a line feed and input line editing functions conform to the rules set forth in the CP/M and MP/M users manuals. By maintaining this degree of compatibility the learning cycle of the AMDS user should be lessened considerably.

System errors are divided into two categories; those due to faulty user inputs and those due to disk I/O operations. Depending on the particular error, errors may take one of three courses of action. They may return directly to the menu, they may restart the option in progress when the error occurred or they may simply return to the point where the error occurred and await user provided corrective measures. More details are provided in the AMDS user's guide.

The final area of the HOST control program requiring discussion is that of the routines and associated protocols used for intercommunication between the HOST and the MTS. Because the MTS may not always utilize a fast processor such
as the Z80 and since the MDS is provided with the ability to execute user programs in real time, it was conceivable that the MDS response time to the HOST could be considerably slow in some instances. This also brings up the possibility of lost data if the HOST is transmitting faster than the MDS can service its serial I/O port. A final problem in such an asynchronous setup is what the data sent is intended for, be it a command or some type of processable data.

In order to alleviate the lost data problem and to lessen the response time to the HOST, several assumptions were made in the communications software design. The primary assumption is that the HOST has communications priority at all times. From this assumption the following protocols were established and implemented. A type of software handshaking between HOST and MDS is provided for each character sent by either device. Some experimentation was done with the use of packets of characters greater than one, but some data loss was experienced when either the HOST or MDS was busy with other tasks besides I/O. Though time prohibited further experimentation in this area, it is felt that some type of hardware initiated control signals would be necessary to increase transmission/reception reliability in a packet communications mode for this system.

The protocol thus implemented follows several rules. For each piece of data to be transmitted two bytes of data are actually required. The first byte indicates the type of
data to follow. Types include command data, pure data, and status data. Each type is assigned a hexadecimal equivalent as follows:

$055H$ indicates that the next byte to be transmitted will be a command

$0FFH$ indicates that the next byte to be transmitted will be pure data

$00H$ indicates that the next byte to be transmitted will be status data (the only currently implemented status data is $00H$, meaning the sender is at some point in the execution of its program where it awaiting input from the other device in order to proceed).

As an example, when the user wants to examine an MDS memory location the HOST first sends the data sequence:

$055H$, $058H$ ($058H$ is the ASCII hexadecimal code for 'X', the Examine Command)

After receipt and display of the data in MDS memory, the user wants to change it to say, $03FH$, thus the HOST would send the sequence: $0FFH$, $03FH$.

In addition to this rule, recall that a software handshake is provided for every character sent. As each character is received, the receiving system returns an acknowledgement byte of $011H$, the ASCII hexadecimal code for XON, meaning the character has been received and further transmissions may proceed. At the same time, the sender is awaiting this acknowledgement before proceeding with further transmissions or continuing on to other tasks. This handshaking overhead seems unrealistically high at first glance, but it is negligible to the user for most types of
applications envisioned for this system and it provides a high degree of confidence in the communications setup. Perhaps the only time the communications throughput would be degraded, in the user’s eyes, would be when an application program might require nearly continuous data transmissions for a lengthy period of time. A way around this particular situation is discussed in the section on system implementation.

To improve MDS response to HOST transmissions, the MDS checks for receipt of a HOST transmission prior to every output to the HOST. If the HOST has sent information, typically a new command, the MDS halts whatever it was doing and processes the new data.

Further details concerning the HOST control program are discussed in the system user’s guide and all routines are well documented in the source code listings and flow diagrams in the appendices.

2. The MDS Onboard Monitor

Because the HOST control program was designed to do most of the the work required of the AMDS, the MDS monitor software was much easier to develop.

The monitor software essentially consists of a command/data interpreter, a set of complementary routines for each of the HOST initiated MDS options, and a similar set of I/O routines for communications with the HOST. The
program flow is basically the same as described for the HOST control program, with the exception that there is no direct input from the user. The MDS monitor does not have any error routines since all system error detection is built into the HOST control program. If for any reason the monitor does not understand the HOST transmissions it simply waits until something is sent that it does recognize and then proceeds. Though it is unlikely that the system will get hung up in a loop during normal HOST to MDS communications, if it should occur, either an ESCape sequence from the HOST or a manual reset of the MDS will terminate the loop. The only foreseeable circumstances in which this might occur are when a user program, executing in MDS memory, attempts to obtain information from the HOST when the HOST is not expecting such a request.

The monitor is written for automatic startup after either a system power-on reset or a manual reset. All MDS serial I/O ports are initialized to communicate at 3600 baud. Routines for user program I/O with the HOST console and for return to the MDS monitor are also provided via simple user calls, as explained in the user's guide.

Again, more detailed information may be best gleened from the AMDS user's guide, the flow diagrams and accompanying source code listings in the appendices.
III. SYSTEM IMPLEMENTATION AND CUSTOMIZATION

The AMDS is a modular system with respect to both software and hardware. Though this thesis is concerned primarily with implementation of the system as already stated, with an ALTOS microcomputer and the PRO-LOG STE hardware, the design is intended to be usable on any other CP/M or MP/M based system with only a few software changes and minor additional hardware interface requirements (beyond the MDS hardware needs, naturally).

A. PUTTING IT ALL TOGETHER

Implementation of the HCST control program is simply a matter of loading and executing the program via the normal CP/M method of typing in the name of the object file, in this case 'AMDS', followed by a carriage return or line feed.

Implementing the MDS system, while not especially taxing, does require the use of a PROM programmer to load the monitor software into EPROM. Once this is accomplished, and the EPROMs are installed, the system implementation is nearly complete. All that remains is connecting the systems together, turning on the power and the reset is automatic.

This particular development system is coupled together via a standard RS-232C connector cable set with a 25-pin,
DB-25P, male 'D' connector on the HOST end and a 26-pin female Amphenol connector on the MIS end. Only the signal ground, transmit and receive signals are necessary and other RS-232C signals are ignored in this implementation. (The standard RS-232C pin definitions are shown in Figure 3.) The HOST end of the connector is plugged into the auxiliary serial port on the ALTCS multi-user system and the MIS end is connected to the 'A' channel socket on the dual USART card. Additionally, it should be ensured that the 'A' channel is jumpered for DTE (Data Terminal Equipment) operation, as explained in the dual USART card documentation listed in the bibliography.

These procedures are all that is necessary to implement and use the basic system.
<table>
<thead>
<tr>
<th>ALTOS</th>
<th>MDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGND</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>NC - No Connection</td>
</tr>
<tr>
<td>RXD</td>
<td>2</td>
</tr>
<tr>
<td>TXD</td>
<td>3</td>
</tr>
<tr>
<td>RTS</td>
<td>4</td>
</tr>
<tr>
<td>CTS</td>
<td>5</td>
</tr>
<tr>
<td>DSR</td>
<td>6</td>
</tr>
<tr>
<td>SGND</td>
<td>7</td>
</tr>
<tr>
<td>RLSD</td>
<td>8</td>
</tr>
<tr>
<td>DTR</td>
<td>20</td>
</tr>
</tbody>
</table>

Protective Ground
Transmitted Data (TXD)
Received Data (RXD)
Request to send
Clear to send
Data Set Ready
Signal Ground
Received Line
Signal Detect
Data Terminal Ready

* NC - No Connection

Figure 3 - RS-232C Pin Definitions and System I/O Setup
B. CUSTOMIZATION

The primary areas of customization of the AMDS are those concerning the use of different processors in the MES and the use of different serial interfaces.

At present the PRO-LOG Corporation STD bus supports the Z80, 6085, Z80, Z80A and the 6600 series microprocessors. The current implementation uses the Z80 with onboard EPROM and RAM. The ROM and RAM address areas may be jumpered to either the lower (as done here) or the upper 16K of address space. In order to use the monitor in the upper 16K of address space would require a hardware addition capable of taking control of the address lines at power-on reset or manual reset, and forcing the next execution address to coincide with the first address of the monitor. Otherwise, the Z80 (and 8080/8085) processors normally execute location 0000H after a reset sequence. If no monitor program is located at this location the processor executes garbage until a HALT instruction is encountered. An implementation of the monitor in high memory, however, is an idea to be well considered for future versions of the AMDS, as it would provide better compatibility with the page zero I/O mapping scheme used by the 6600 microprocessor. As an additional benefit, it would lessen some of the software limitations currently imposed by the current configuration. These limitations are discussed in a separate section of this paper.
As to the use of different serial I/O interfaces, several hardware additions may be necessary on the ALTOS computers. If the system is used with the single-user ALTOS computers, the options are to use the serial port currently used by the printer or to build an additional serial port into the computer via the use of its internal bus connector. If using the multi-user system, two AMDS systems could be supported simultaneously by simply using two of the serial ports currently used for consoles. To support four complete AMDS systems would require the addition of three more serial ports in a manner similar to that discussed for the single user system.

The changes in serial port usage would require a few minor changes in the HOST control program. If ZILOG SIO devices are used, as presently installed in the ALTOS series computers, the software modification reduces to simply changing the status (MSTATPT) and data (MDATAPT) port designations in the 'equates' (EQU statements) section at the beginning of the HOST control software source code and then reassembling the code for the new serial ports. If serial communication chips other than the SIO are used, the HOST control routines MDSTAT, ME SIN, and MESOUT would have to be modified to operate with the particular chip chosen.

On the MDS side of the system, the customization process for software changes of serial ports is very similar to that of the HOST. Using additional INTEL 8251 USARTs would
necessitate only changes to serial port equates for CHASTAT and CHADATA in the MDS monitor source code, followed by reassembly and reprogramming of the EPROMs. Use of serial devices other than the 8251, would require appropriate changes to the MDS routines HOSTAT, HOSTIN, and HOSTOUT.

Beyond these hardware oriented customization procedures, provisions have been included for the addition of more user options and error processes in the HOST control software. Each of these areas use 'jump' tables to vector to the option or error routine selected. To add an option to the menu, the new option routines would be added to the body of the current source code, a JMP xxxx (xxxx is the option label) instruction would be added to the menu jump table and the menu display would be modified appropriately in the message storage section of the source code. The insertion of additional error codes is identical, except that the jump instructions are inserted in the error jump table.

One further comment on the addition of user options concerns the method of decoding the option selected. Menu options are identified by an assigned alphabetic character from A through Z (current options go only through the letter N). The ASCII code for each option is modified for use with the jump table in the following manner. The ASCII code is first 'anded' with the data 01Fh. This removes all ASCII biasing and leaves only the hexadecimal equivalents of the numbers 1 through 26, corresponding to the letters A to Z.
These numbers are then used to find the appropriate vector from the jump table, as further explained in the source documentation. Thus the provision for twelve more options, 0 through Z, is included in the current version of the HOST control software. If these options are added, simple changes are also required to the equates for MAXCHCE, the highest option letter in use, and for NHSTCMCL, the current number of 'host only' commands.

A consideration to keep in mind when editing the HOST software is the fact that it is currently a 62K byte file and thus larger than the index table capacity of the TEE text editor used widely at the Naval Postgraduate School. For this reason, the source code is broken into two files: AMLSP1.ASM containing the primary option routines, and AMLSP2.ASM containing the utility and support routines and message and data storage definition areas. Prior to assembly, the files are concatenated via the use of the CP/M Peripheral Interchange Program (PIP) as follows:

```
PIP AMDS.ASM=AMDSPI.ASM,AMDSF2.ASM
```

The file AMDS.ASM is then assembled using whatever assembler is desired.

MON monitor software customization is at least as simple, if not easier than that for the HOST. Commands are decoded via the simple mechanism of comparing the command to a set of known commands and then jumping to the option...
routines selected. The only additional source code changes which might be applicable to the MIS would be a change of the assembly origin (ORG statements) addresses if the monitor is to be moved into upper memory as mentioned previously.

C. SYSTEM LIMITATIONS

This system, as with many other well designed systems, also has its limitations. Some of these have already been alluded to in previous sections and will now be discussed in more depth.

The current MDS configuration, with the lower 16K address space reserved for the monitor ROM and RAM, imposes several notable limitations on the use of the AMDS. Besides the page zero I/O mapping incompatibility between the E800 and Z80, which has already been pointed out, the inability to use this address space for user program execution places a restriction on the types of CP/M based software which may be downloaded and executed in the MIS memory.

CP/M's executable object files, designated as '.COM' files, are created with the implied intent of loading and initiating the execution of these files from location 0100H. Since this location is within the reserved area in the MIS, such '.COM' files cannot be downloaded and executed in MIS memory. Unfortunately, most CP/M software on the commercial market is distributed in this format.
The restriction thus imposed is that only disk files in the INTEL Hex Format (see Figure 4) or in a page relocatable format may be downloaded and executed in MDS memory. This is because these formats are not dependent upon any address restrictions and are executable in whatever address space for which they are assembled.
RH - RECORD HEADER: AN ASCII COLON (3A HEX) SIGNALS THE START OF EACH RECORD.

RL - RECORD LENGTH: TWO ASCII HEX CHARACTERS GIVE THE RECORD LENGTH (THE NUMBER OF 8-BIT DATA BYTES IN THE RECORD). END OF FILE IS INDICATED BY A ZERO RECORD LENGTH. (10 HEX IS MAX. RL)

LA - LOAD ADDRESS: FOUR ASCII HEX CHARACTERS GIVE THE ADDRESS WHERE THE FIRST DATA BYTE OF THE RECORD IS LOCATED.

RT - RECORD TYPE: THE RECORD TYPE IS ALWAYS 00 EXCEPT FOR THE LAST RECORD OF AUTOSTART FILES, WHERE IT IS 01.

DATA - TWO ASCII HEX CHARACTERS REPRESENT EACH 8-BIT DATA BYTE.

CK - CHECKSUM: TWO ASCII HEX CHARACTERS GIVE THE NEGATIVE SUM OF ALL PREVIOUS BYTES IN THE RECORD, EXCEPT FOR THE COLON. THE SUM OF ALL THESE BYTES PLUS THE CHECKSUM EQUALS ZERO.

Figure 4 - INTEL HEX File Record Format
The free address space of the present MDS, 4000H to 0BFFFH, is therefore sufficient for the needs of these file types. As mentioned, most distributed software does not come in these formats. For use of the MDS in beginner and intermediate level course work, however, this restriction should not be a dominant disadvantage in applications development and in gaining an insight into the use of microprocessors.

Because of the time constraints imposed, as well as this student's lack of familiarity with page relocatable file formats, only the use of type '.EEX' files are supported for upload and download operations in the current version of the AMDS.

Other limitations of the system are: the lack of breakpoint setting and cpu register examination facilities in the MDS; the lack of a facility for moving blocks of MDS memory; the inability to operate the MDS in a true in-circuit emulation mode; the current limitation of having only a single processor and the inability to operate multiple processors on the MDS bus; and the limitations already discussed concerning communications protocols.

Most of these limitations are only temporary, with the possible exception of obtaining true in-circuit emulation. The high communications overhead of the HOST to MDS interface can be avoided by user programs in the MDS memory.
simply by utilizing a separate console and the additional MDS serial port when the need for high speed data transfer arises.
IV. CONCLUSIONS AND RECOMMENDATIONS

The original needs stated for the microprocessor development system have been met, with the exceptions noted as limiting factors. Even with these limitations imposed on the current design, however, it is felt that a significant tool has been added to the small, but growing Electrical Engineering microcomputer laboratory. The final design of the system has left considerable room for future expansion and improvement in both areas of software and hardware and is thus a good vehicle for additional thesis study.

A. FUTURE HARDWARE

There are numerous changes and enhancements to be made to the system in the hardware area. Some of these enhancements are described below.

Implementation of hardware initiated communication control signals to increase system response and throughput.

The addition of a Master/Slave cpu capability to operate and evaluate different microprocessor types on the same bus; this capability would have to be implemented via the use of interrupts and the bus request control lines plus appropriate software.

The addition of analog to digital and digital to analog (A/D and D/A) capability will significantly increase the usefullness of the system in hardware development applications.
Another worthwhile improvement would be the addition of a PROM programmer with the capability to change its personality under software control in order to program different types of PROMs.

..... and the list goes on.

B. FUTURE SOFTWARE

Many of the immediate enhancements to the system will probably be an outgrowth of the limitations pointed out previously. These include making changes for the use of CP/M `.COM' files and adding support for page relocatable files. These two additions alone, would tremendously improve the potential uses of the AMDS.

Other near future additions should include facilities for moving blocks of MDS memory and for the use of breakpoint, single-stepping and program trace routines. Such routines would probably be best implemented as individual files downloaded to the MDS memory. The routines could then operate as an extension of the onboard monitor. This would also provide the flexibility to execute routines for different processors under control of a dedicated monitor.

The addition of software for cross assembly of source code between various processors is another recommendation worth careful consideration. One idea, which was considered for inclusion in this thesis but was axed for lack of time, is the use of macro assemblers for cross 'translation' of source code. The idea would be to develop source code using
the standard mnemonics of a particular processor and then translate the source code to the mnemonics understood by whatever processor is actually available. Once this is accomplished, testing and debugging of the software can be done with available hardware. The code can then be translated or cross assembled back to code for the original processor and put to use in its intended application, all without the use of a true development system for that processor.

Finally, an area of great promise is that of systems networking. The new CPNET and MPNET loose-coupled network facilities, by DIGITAL RESEARCH Corporation, provide numerous avenues for further study into allowing the AMDS to share its resources with other computer systems.

All of these improvements are feasible and cost effective. These additions will also allow much of the burden to be taken off the beginning program and hardware designers. Much of the less interesting trivia normally associated with applications development can be skipped over and the solution to the problem can be approached in a more efficient and structured manner.
## APPENDIX A

**AMDS USERS GUIDE**

### TABLE OF CONTENTS

1. **Introduction** .................................................. 40
2. **How to Use the AMDS** ........................................ 41
3. **Getting Started** ............................................... 43
4. **System Functions (User Options)** .......................... 44
5. **Information of General Interest** ......................... 52
6. **Tips for Programming the MIS** .......................... 54
7. **System Error Messages** .................................. 57
1. INTRODUCTION

The ALTOS Microprocessor Development System (AMES) is designed to be used as an aid to students in beginning and intermediate levels of software and hardware applications development. The system consists of an ALTOS microcomputer, running under the CP/M or MP/M operating systems, and a hardware development and testing system built around the PRO-LOG STD bus. Included in the current (June 1981) hardware development system are a 2MHz Z80 cpu card with onboard monitor in EPROM and 4K bytes of static RAM, two 16K byte static RAM cards and a dual UART asynchronous RS-232C serial I/O card. The ALTOS and the hardware development system are linked together via a serial I/O channel.

The ALTOS computer, hence referred to as the 'HOST', exercises control over the hardware development system (designated as the 'MDS') via the execution of the HOST control program named AMES.COM. The onboard monitor in the MDS contains routines which complement those in the HOST control program, though on a less complex scale. A more detailed treatment of the inner workings of the AMD3 system is available in the student thesis by LT. Stephen M. Hughes, USN, titled "A Microprocessor Development System for the ALTOS Series Microcomputers".

40
2. HOW TO USE THE AMDS

The AMDS' primary use is in the design and testing of both software and hardware applications in a real time environment. The typical steps for effective use of the system would be as follows:

a) Using standard CP/M or MP/M software development tools, such as LDT, TEL, ED, ASM and MAC, the user would develop, test and debug (to the extent possible) software to be used in a hardware/microprocessor oriented application.

b) Simultaneously to step a), the user, or other members of a project team, would be designing, wire wrapping and performing initial tests on the hardware, using available test equipment such as oscilloscopes, digital voltmeters, etc.

c) At such time as the hardware and software are ready to be tested together, the AMDS would come into use. At this point the wire wrapped circuitry would be inserted into a slot in the development bus, the software would be downloaded to the MES memory and, via the use of the AMDS user options, the software and hardware would be tested as a single unit.
d) Refinements and correction to both hardware and software could then be made as in steps a) and b) and step c) then repeated until the application operates as intended.

The intent of this procedure, though it might appear cumbersome, is to allow the software programmers to concentrate on their work using proven and tested development aids while simultaneously allowing the hardware designer/builders to forge ahead in their respective areas. The lesson to be learned is the 'real world' concept that communications between such distinct but collectively important segments of a team effort are what is necessary for successful fulfillment of the project goals. These intergroup communications require that each team carefully plan the project in its initial stages of development and that the division of responsibilities and the methods of implementation of the project are thoroughly understood by all members of the team. With this type of planning and communication of ideas, the AMES concept is thus seen as less cumbersome than initially thought and actually allows for a very flexible working environment. The use of the AMES also relieves the hardware designers of much of the burden previously placed on students to design and wire wrap their own cpu and memory cards.
3. GETTING STARTED

This section is intended as a quick review for those already familiar with the use of the AMDS. Others should carefully review the remainder of this guide prior to attempting to use the system.

With software developed and tested as best possible (naturally those software routines fully dependent upon the hardware have not been completely tested) and with the hardware prototype in hand, the stage is set for utilization of the AMDS.

With the MLS power OFF (!) the prototype card is inserted snugly into one of the wide slots of the card cage which are specially designed to accept wire wrapped cards. After insuring the card is properly in place, the power is then switched on and the MLS reset switch is pressed. The MLS is now ready for use.

Next, the AMDS HOST control software is initiated from the ALTOS system console by typing ‘AMLS’, followed by a carriage return. The HOST control program then loads into memory and begins execution by displaying a menu of user options and prompts the user for a reply. At this point the user(s) may proceed with testing using the options described in subsequent sections of this guide.
4. SYSTEM FUNCTIONS (USER OPTIONS)

The AMDS control program is designed as a menu-driven program. This means that after each primary task is completed, the user is shown a menu of options from which he may choose his next move. Each of these options is discussed in the remainder of this section of the guide.

A. SUPPRESS PRINTING MENU -
Selection of option 'A' allows the experienced AMDS user to automatically suppress the display of the menu at the end of each option. When this is done the system status (whether the HOST or MDS is in control) and reminders of which option suppresses and which does not suppress the menu are printed, followed by the prompt to input a menu option.

B. DO NOT SUPPRESS PRINTING MENU -
Opposite of option 'A', option 'B' allows the user to regain full menu display if he cannot remember the option code he wishes to select.

C. BASIC INSTRUCTIONS -
Option 'C' displays a set of basic instructions for use of the AMDS. These instructions should normally answer the questions of most first time users without the need to resort to this guide.
D. HEXADECIMAL ADD and SUBTRACT -
Option 'L' allows the user to quickly obtain the 16 bit hexadecimal sum and difference of two numbers. When this option is selected, a message verifying the option actually entered will be displayed, followed by a prompt for input.

The input expected is two hexadecimal numbers, of up to four digits each, separated by either a comma or a space as the following example shows:

>01AF,F3AB or >01AF,F3AB

The sum and difference of these two numbers are then displayed as:

SUM = F55A  LIFF = 0E04

The user is then returned to the menu for selection of another option.

( ** This option has the same input format as the 'H' command in EET ** )

E. RETURN SYSTEM CONTROL TO HOST -
Selection of option 'E' is necessary only when the system control has been passed to the MDS via a previous command for it to execute a program in its own memory. This option then allows the user to request the MDS to
terminate its present action and return control to the HOST in preparation for subsequent commands.

** Note that this option may not be effective if the program being executed in MLS memory runs astray or never checks for or attempts to perform I/O with the HOST. The only remedy in this situation is to manually reset the MDS.

F. RETURN TO CP/M –
Selection of option ‘F’ will terminate use of the AMDS and return the user to the CP/M (or MP/M) operating environment. (The input of a control C as the first entry after any prompt will also accomplish the same thing.)

G. DOWNLOAD HEX FILE – DISK TO MLS –
Option ‘G’ allows the user to download an INTEL Hex format file from disk to MLS memory. Hex files are normally generated in the course of the assembly process.

** Note that only ‘HEX’ file types are supported in this version and the system will not accept requests for any other types.

When this option is selected, an option verification message is displayed and the user is prompted to input the filename. The entry of the filetype ‘HEX’ is
optional but acceptable. Rules for acceptable filenames follow those set forth in CP/M documentation with the exception that ambiguous filenames (those containing ?'s) are not accepted. Additionally, only the currently logged in disk drive will be used for disk I/O and if the drive select code is entered with the filename it will be ignored if it fails to match that which is currently logged in.

After the Hex file is successfully downloaded, a message to that effect will be displayed and the user will be returned to the menu.

H. UPLOAD MIS MEMORY TO HEX DISK FILE -
Option 'H' is just the reverse of option 'G'. Filename input is the same. After the filename is input, the user is prompted for the starting and ending addresses in MIS memory from which the contents are to be saved on disk in a 'HEX' type file. Acceptable inputs are two hexadecimal numbers, the first being less than the second, input in the same manner as in option 'L':

>403C 659F

When the upload is completed, the user will be so informed and returned to the menu.
I. EXAMINE/SET MDS MEMORY LOCATION(S) -
Option 'I' allows the user to examine and modify (set) the contents of MDS memory. The first prompt is for the initial MDS address to be examined such as: >0B03. The system then fetches the data from that location and displays it as:

0B03 3A

and waits for more input after the '3A'. If the user desires to change the data in that memory location, he may then enter the new data. The system stores the new data and automatically advances, examines and displays the next sequential location in MDS memory. This process continues until a period is the only data input. If no modification of a memory location is desired, a carriage return will cause an advance to the next memory location without modifying the MDS memory.

( ** This option has the same I/O format as the 'S' command in DET ** )

J. CONTINUOUS SET OF MDS MEMORY -
Option 'J' is similar to the examine/set option ('I') except that it does not examine the MDS memory, it only modifies it with sequential input data. The first input requested is the starting MDS address for modifications, i.e. >13DA. The second and subsequent prompts are for
data to be entered into MDS memory, sequentially starting at the address specified. Input data may be up to 255 characters long (including spaces and commas) for a single line of input. If more than 255 characters are input, the system merely issues another prompt for a continuation line. Each byte of data is separated by a space or a comma. When input is completed, a period entered after the prompt will terminate the option.

K. FILL MDS MEMORY WITH SPECIFIED BYTE - Option 'K' enables the user to fill any portion of MDS memory with a specified byte of data. The advantage of this is to allow the user better knowledge of the current contents of MDS memory and to help in identifying needed data during memory dumps to the CRT. The input expected after the prompt are the start and ending MDS addresses followed by the data to be placed in those locations. For example:

>0395,7FD0,2A will fill MDS memory between, and including, locations 0395H and 7FD0H with data 2A, the ASCII code for '*

( ** This option has the same input format as the 'F' command in LRT ** )

49
L. LOCATE BYTE SEQUENCE IN MLS MEMORY -

Option 'L' allows the user to search MLS memory for a sequential data sequence up to 16 bytes long. The first input prompted for is the search start address followed by an optional end address as shown:

>0023 579A or >023

If no end address is given it will default to 0FFFFH. The next prompt is for the byte sequence as:

>00 03 45,9A,CC ..... up to 16 bytes

If the sequence is found, the starting address of the sequence in MLS memory is displayed. If not found, an appropriate message is also displayed.

M. DUMP MLS MEMORY LOCATION(S) TO CONSOLE -

Option 'M' provides for a hexadecimal and ASCII MLS memory dump to the CRT. The only inputs required are the start and optional end addresses for the dump in the same format as option 'L'. If no end address is specified it defaults to the start address + 256.

( ** The dump I/O format is the same as that for the 'D' command in DET ** )

If the user wishes to continue the dump after the initial dump completes, he may type in the letter 'L' to
dump the next 256 byte block. Any other input will return the user to the menu.

** Note that unlike the LIT dump command, the only way to abort a memory dump is by pressing the ESCape key.

N. EXECUTE MLS MEMORY FROM A SPECIFIED LOCATION - Option 'N' allows the user to pass system control to the MLS and let it execute a program in its memory. User input required is the MLS start address of the program to be executed. After the address is input, the user is asked whether or not the program to be executed in MLS memory will be sending data to the HOST console for display. If the answer is no, then the user is returned to the menu. IF the answer is yes, then the HOST system loops waiting for data to display, until one of the conditions mentioned below is met.

** Note that when this option is selected, the options F through N are disabled until the MLS returns control to the HOST; when the 'E' option is selected; or when the MLS system is manually reset.

** For further discussion on the proper use of this option, see the section on 'TIPS FOR MLS PROGRAMMING'.
5. INFORMATION OF GENERAL INTEREST

a) The prompt for all user input is ‘>’.

b) All inputs may be in either upper or lower case alphabets.

c) All input is terminated with either a carriage return or a line feed.

d) All address and data inputs are expected to be in hexadecimal notation. Address inputs contain from 1 to 4 hex digits and data inputs contain 1 or 2 hex digits.

e) When inputting addresses and data, mistakes may be corrected in two ways: 1) by using the RU/EOUT key or backspace keys to delete input or 2) by simply continuing to input the hex characters until the correct ones are input. For addresses, the program always takes the last four or less hex digits input and for data, the last two or less digits entered. At least one digit must be entered for every required input parameter.

f) A question mark ‘?’ entered during input will cause the required input formats for each option to be displayed. When the display is completed, the currently selected option is restarted.
g) If the ESCape key is entered as input, the option is immediately terminated and the user is returned to the menu.

h) The MDS is automatically reset at power-on but it is generally a good idea to manually reset it anyway.

i) The MDS to HOST serial I/O port and the additional I/O port in the MDS are both initialized at every reset to operate at a 9600 baud rate.
6. TIPS FOR PROGRAMMING THE MDS

a) If a program requires considerable communications with the user, the best terminal response will be gained by using a separate CRT attached to the spare serial I/C port in the MDS. This port may be reprogrammed for a different baud rate if necessary (see the PRO-LOG dual UART documentation for detailed steps for programming channel B).

b) If the user does not wish to fool with programming the MDS channel B USART, but still has the need for console I/O, his program may use the routines built into the monitor specifically for this purpose. In a manner similar to the BIOS calls used by CP/M, the user program may call location 0005H in the monitor for console I/O using the HOST console. The conventions for these calls is as follows:

- for input from the HOST console the user program should call MDS address 0005H with the function code 01H in register C; the character from the console will be returned in the Accumulator

- for output to the console, a call is made to MDS address 0005H with the function code 02H in register C, and the character for output in the Accumulator
- to merely check to see if input has been received from the HOST, address 0005H is called with function code 03H in register C; if no character is waiting the accumulator will be returned = 00H, otherwise A = OFFH meaning input has been received.

- if a call is made to MDS address 0005H with a function code in register C other than 01H, 02H or 03H, no I/O will take place and the C register will be returned with OFFH.

**Two points should be remembered when using the HOST console for I/O:

1) the data returned from the I/O port is a full eight bits as received with no stripping of the high order bit for ASCII data.

2) when the console is to be used for user program I/O, be sure to answer yes to the query about console I/O when option 'N' is selected.

c) if no I/O with the host console is necessary, as in a) above, the user program should at least periodically check the HOST port status to see if it wants to terminate the execution of the user program. If data is waiting a call should be made as explained above to fetch the data so that the monitor can interpret it.
d) the user always returns control to the HOST via a jump to location 0038H in MDS memory; a RST 7 instruction will also accomplish the same thing.

e) do not forget that MDS user memory starts at location 4000H and all HEX files should be assembled for addresses above that location.
7. SYSTEM ERROR MESSAGES

System error messages are the result of either user data input errors or disk I/O errors. A list with brief explanations follows:

A. USER INPUT ERRORS –

INVALID MENU SELECTION – this message is displayed when an option is input which is not one of the selections from the menu. (* this error returns the user to the menu *)

TOO MANY OR TOO FEW DELIMITERS IN INPUT – used to indicate that too many or too few parameters were input than expected. Acceptable delimiters are a space or a comma. (* this error restarts the current option *)

PERIOD ONLY PLEASE ! – given when a period is input to terminate input and the period is preceded or followed by other input data. Only a period may be input. (* this error restarts the current option *)

INVALID HEX DIGIT – an input of a non-hex digit (not in the range 0-9, A-F) was attempted. (* this error restarts the current option *)

57
CAN'T HAVE A DELIMITER AT START OR END OF INPUT -
either a space or a comma was input as the first or last
character in an input line. (* this error restarts the
current option *)

TWO OR MORE DELIMITERS SEQUENTIALLY - too many
delimiters were inserted between input parameters. (*
this error restarts the current option *)

AMBIGUOUS FILENAMES NOT ALLOWED - the filename which
was input contained a '?' . (* this error reprompts for
new input *)

COLON (:) NOT PROPERLY PLACED IN FILENAME - the only
colon allowed in the filename is after the drive code
and before the first letter of the filename. (* this
error reprompts for new input *)

FILENAME TOO LONG OR TOO SHORT - maximum filename
length is 8 characters; minimum is 1. (* this error
reprompts for new input *)

HEX FILETYPES ONLY! - only files of type '.HEX' are
implemented in this version. (* this error reprompts for
new input *)
NO SPACES ALLOWED IN FILENAME - filename characters must be sequential with no spaces. (* this error reprompts for new input *)

NON-PRINTABLE CHARACTERS NOT ALLOWED IN FILENAME - only printable characters are allowed in filename. (* this error reprompts for new input *)

START ADDRESS CANNOT BE GREATER THAN FINISH ADDRESS - when in the UPLOAD option, the user must specify MIS memory address boundaries for upload with the start address lower than the end address. (* this error restarts the upload option *)

WARNING - ONLY CURRENTLY SELECTED DISK WILL BE USED, INPUT IGNORED ! - this version of AMDS does not allow disk drive specification unless it is the same as the disk currently logged in to the user. Other drive specifications are ignored and the option defaults to the currently logged disk.

B. DISK I/O ERRORS -

FILE NOT FOUND - the file specified cannot be found in the directory for download to the MIS. (* this error restarts the download option *)
HEX CHECKSUM ERROR - a data error was detected while trying to download a HEX file. (* this error returns the user to the menu *)

DISK READ ERROR - an attempt was made to read a disk file but was unsuccessful; check diskette media then the disk drive. (* this error returns the user to the menu *)

OUT OF DIRECTORY SPACE - disk directory is full; delete files or use another diskette. (* this error returns the user to the menu *)

OUT OF DIRECTORY OR DISK STORAGE SPACE - ran out of space in one of these areas while attempting to write data to a disk; *** when this occurs, the data already written is deleted, i.e. NO PARTIAL files are saved ***. (* this error returns the user to the menu *)
APPENDIX B

FLOWCHARTS FOR HOST AND MDS USER OPTIONS

OPTION A
- SET MENU SUPPRESSION FLAG
- MENU

OPTION B
- RESET MENU SUPPRESSION FLAG
- MENU

MENU SUPPRESSION

OPTION C
- PRINT INST'S
- MENU

OPTION D
- GET 2 NO'S CVT TO BIN
- DO ADD & SUBTRACT
- CVT TO HEX & DISPLAY
- MENU

BASIC INSTRUCTIONS

HEX ADD/SUBTRACT
OPTION E

REQUEST CONTROL FROM MDS

REQ. ACK?

YES NO

MENU

RETURN CONTROL TO HOST

OPTION F

PRINT SIGNOFF

CP/M(MP/M)

RETURN TO CP/M

RETURN TO HOST
DOWNLOAD HEX FILE TO MDS MEMORY

OPTION G

GET FILENAME

OPEN FILE

FILE FND?

YES

SEND MDS DWNLD CMD

READ A HEX RECORD

SEND RL, LA & DATA TO MDS

EOF?

NO

YES

TELL MDS DONE

MENU

(HOST FLOW)

DWNLD

GET RL, LA, DATA STORE

END DWNLD?

NO

YES

MONITOR

(MDS FLOW)

63
UPLOAD FROM MDS MEMORY TO HEX DISK FILE

- OPTION H
- GET FILENAME
- FILE EXIST?
  - YES DELETE FILE
  - NO CREATE NEW FILE

(HOST FLOW)

- UPLD
- GET START & END ADDRS.
- SEND DATA TO HOST
- DONE?
  - NO
  - YES

(MDS FLOW)

- GET START & END ADDRS.
- SEND UPLD CMD & ADDR TO MDS
- GET DATA FROM MDS
- FORMAT & WRITE TO DISK
- END XFER?
  - NO
  - YES
   - CLOSE FILE
   - MENU

- TELL HOST DONE
  - MONITOR

END NO
EXAMINE/SET MDS MEMORY

HOST FLOW:

1. **OPTION I**
   - GET START ADDRESS
   - SEND CMD & ADDR to MDS
   - GET DATA FROM MDS
   - DISPLAY DATA

2. GET NEW DATA
   - IS IT \( '.', ' \)?
     - YES
     - SEND DATA TO MDS
     - TELL MDS DONE
   - NO

MDS FLOW:

1. EXAM
   - GET START ADDRESS
   - SEND DATA TO HOST
   - DONE?
     - NO
     - GET NEXT MDS DATA
     - MONITOR
   - YES
CONTINUOUS MDS MEMORY SET

OPTION J

GET START ADDRESS

SEND CMD & START ADDR

GET DATA FOR MDS

IS IT '...'? YES

NO

SEND DATA TO MDS

TELL MDS DONE

MENU

GET START ADDRESS

GET DATA FROM HOST: STORE IT

DONE? NO

YES

MONITOR

(MHOST FLOW) (MDS FLOW)
FILL MDS MEMORY WITH SPECIFIED BYTE

OPTION K

GET START & END ADDR'S & DATA

SEND CMD + ADDR's/DATA

MDS DONE? NO

YES

MENU

FILL

GET ADDR'S & DATA

FILL MEMORY

DONE? NO

YES

TELL HOST DONE

MONITOR

(HOST FLOW) (MDS FLOW)
LOCATE BYTE SEQUENCE IN MDS MEMORY

**HOST FLOW**

1. **OPTION L**
   - GET START/END ADDR'S & DATA
   - SEND ADDR'S & DATA/CMD
   - SEQ. FOUND?
     - NO
     - PRINT FOUND ADDR
     - YES
     - MENU

**MDS FLOW**

1. **LOCATE**
   - GET ADDR'S & DATA
   - DO SEARCH
   - SEQ. FOUND?
     - NO
     - SEND NOT FOUND MSG
     - YES
     - SEND FOUND MSG
     - MONITOR
DUMP MDS MEMORY TO THE HOST CONSOLE

HOST FLOW

MDS FLOW
EXECUTE USER PROGRAM IN MDS MEMORY

**HOST FLOW**

1. OPTION N
   - GET EXEC. ADDRESS
   - SEND CMD+ ADDRESS
   - MDS I/O?
     - YES: PERFORM I/O
     - NO: MENU

**MDS FLOW**

1. EXEC
   - JUMP TO EXEC ADDR.
   - MDS EXEC. COMPLETE
   - MONITOR
APPENDIX C

AMES HOST CONTROL SOFTWARE LISTING

**************************************************************************************
* AMDS - ALTOS MICROCOMPUTER DEVELOPMENT SYSTEM
* (HOST CODE)
* VERSION 1.5, 28 MAY 1981
* LT. STEPHEN M. HUGHES - author
* This is the HOST (ALTOS) control code for the AMDS.
* Separate code for the MDS onboard monitor is listed
* under the filename AMDS1.ASM.
* The AMDS user's manual should be consulted for
* specifics not given in the documentation which follows.
**************************************************************************************

org 100h

CPM EQU 0000H ; WARM BOOT RE-ENTRY TO CP/M
EDOS EQU 0005H ; EDOS ENTRY POINT
MSTATPT EQU 25H ; I MDS SIO STATUS PORT
MDATAPT EQU 28H ; I MDS SIO DATA PORT
CONIN EQU 1 ; CONSOLE INPUT FUNCTION
CONOUT EQU 2 ; CONSOLE OUTPUT FUNCTION
PRTSBG EQU 9 ; PRINT STRING TO CONSOLE
READCON EQU 10 ; READ CONSOLE BUFFER
CONST EQU 11 ; CONSOLE STATUS FUNCTION
OPENF EQU 15 ; OPEN FILE FUNCTION
CLOSEF EQU 16 ; CLOSE FILE FUNCTION
DELF EQU 19 ; DELETE FILE FUNCTION
READF EQU 20 ; READ SEQUENTIAL FUNCTION
WRITEF EQU 21 ; WRITE SEQUENTIAL FUNCTION
MAKEF EQU 22 ; MAKE FILE FUNCTION
CURRTND EQU 25 ; GET CURRENT DISK FUNCTION
SETDMA EQU 26 ; SET DMA ADDRESS FUNCTION
CR EQU 0DH ; ASCII CARRIAGE RETURN
LF EQU 0AH ; ASCII LINE FEED
ESC EQU 1Bh ; ASCII ESCAPE CODE
COMMA EQU , ; ASCII COMMA
PERIOD EQU . ; ASCII PERIOD
SPACE EQU \ ; ASCII SPACE
BKSPE EQU 08H ; ASCII BACK-SPACE
XON EQU 011H ; CONTROL Q
MINCHCE EQU 'A' AND 1FH ; MINIMUM MENU CHOICE
MAXCHCE EQU 'N'+1 AND 1FH ; MAXIMUM MENU CHOICE
EOP EQU 1AH ; CONTROL Z - END OF FILE CT
          ; BUFFER INDICATOR
NHSTCMD EQU 6 ; CURRENT NUMBER OF HOST CLS
STACK EQU $ ; 64 LEVEL STACK AVAILABLE

STARTER XRA A ; INITIALIZE HOST IN CONTROL.
STARTA SYSSTAT
STARTA MENUSUPF
LXI D, SIGNON ; PRINT SIGNON AND BASIC INSTRUCTIONS
CALL PRINT

MENU XRA A
STARTA MDSRDYF
INR A ; DEFAULT TO NO MENU
STARTA MENUSFLG ; SUPPRESSION ON MENU ERRORS OTHER THAN INVALID CHOICE
MVI A, 48 ; INIT. CONSOLE READ BUFFER
STARTA CONBUFF ; TO 46 CHARACTERS MAX
LXI SP, STACK ; SET STACK POINTER
LLA MENUSUPF ; PRINT MENU?
ORA A
JNC XRA
LXI D, MFMELERR ; INPUT ERROR (SYNTAX LIKELY)
CALL PRINT

MENU01 CALL STATSYS ; DISPLAY SYSTEM STATUS
CALL BUFFRD ; GET MENU CHOICE
XRA A ; NO DELIMITERS ALLOWED
CALL SCAN ; CHECK INPUT FOR DELIMITERS
JNC MENU011 ; Scan OK
LXI D, MFMELERR ; INPUT ERROR (SYNTAX LIKELY)
CALL PRINT
CALL DELAY ; DELAY TO READ ERROR MSG
JMP MENU ; BACK TO MENU

MENU011 INX D ; ALL INPUT OK, POINT TO IT
LCR B ; AT END OF BUFFER YET?
JNZ MENU011 ; NO, TRY AGAIN
LDA D ; GET OPTION
ANI 1FH ; DELETE ASCII EIAS
CPI MINCHCE ; IS CHOICE < 'A'?
JC MENU012 ; YES, ILLEGAL CHOICE
CPI MAXCHCE ; IS CHOICE VALID?
JC MENU013 ; APPEARS TO BE

MENU012 MVI A, 1 ; NO - PRINT ERROR MSG #1
JMP ERROR

MENU013 PUSH PSW ; SAVE OPTION
CPI NHSTCMD ; IF HOST CMD, MDS CONTROL
JC MENU014 ; HAS NO EFFECT (EXCEPT
                ; EXIT CMD)
LDA SYSSTAT ;GET SYSTEM STATUS
ORA A
JZ MENU014 ;HOST IN CONTROL
LXI D,CNTRLMSG ;MDS IN CONTROL
CALL PRINT
JMP MENU ;ONLY ESCAPE WILL GET
            ;CONTROL BACK
MENU014 POP PSW ;RETRIEVE OPTION
MENU1 STA MENUFLG ;SAVE CHOICE FOR USE IN
            ;HELPING USER LATER
CALL MENUCH ;BRANCH TO APPROPRIATE
            ;CHOICE

MENUCH MOV C,A ;COMPUTE MENU CHOICE VECTOR
MVI B,0
LXI H,CHOICE-3
DAD B
LAL B
DAD B
PCHL ;CHOICE VECTOR IS IN PC
NOP
NOP

* THIS JUMP TABLE MAY BE ADDED TO FOR FUTURE EXPANSION UP *
* TO 26 MENU CHOICES *

CHOICE JMP MENSUP ;SUPPRESS MENU
JMP NOMENSUP ;DO NOT SUPPRESS MENU
JMP INST ;INSTRUCTIONS
JMP HEXARITH ;HEX SUM & DIFF.
JMP RCNT2HST ;RETURN CONTROL TO HOST
JMP CPM ;RETURN TO CPM

* MDS COMMAND JUMP TABLE *

JMP DWNLD ;DOWNLOAD HEX FILE
JMP UPLD ;UPLOAD HEX FILE
JMP EXAM ;EXAMINE/SET MDS MEMORY
JMP CSET ;CONTINUOUS SET W/O EXAMINE
JMP FILL ;FILL MDS MEMORY
JMP LOCATE ;LOCATE BYTE SEQUENCE IN
            ;MDS MEMORY
JMP DUMP ;DUMP MDS MEMORY
JMP EXEC ;EXECUTE MDS MEMORY

*** HOST COMMANDS ONLY - MDS DOESN´T CARE WHAT IS ***
*** HAPPENING ***

* MENU SUPPRESSION *
MENSUP MVI A,1 ;SET MENU SUPPRESSION FLAG
STA MENUSUPF
JMP MENU

* NO MENU SUPPRESSION (DEFAULT) *

NOMENSUP XRA A ;RESET MENU SUPPRESSION FLAG
STA MENUSUPF
CALL CRLF
JMP MENU

* INST - INSTRUCTIONS *

INST LXI L,INSTRUC ;PRINT INSTRUCTIONS
CALL PRINT
INST1 CALL CONSTAT ;WAIT FOR RESPONSE
RRC
JNC INST1 ;LOOP
CALL CONSIN ;GET CHARACTER
JMP MENU

* HEXARITH - ADDITION/SUBTRACTION OF TWO HEXADECIMAL *

HEXARITH LXI L,HEXMSG ;PRINT VERIFICATION MESSAGE
CALL PRINT
CALL BUFFRD ;GET INPUT
MVI A,1 ;ONE DELIMITER REQUIRED
CALL SCAN ;CHECK FOR IT
JNC HEXI ;ALL DELIMITERS OK
MVI A,2 ;DELIMITER ERROR
JMP ERROR
HEXI CALL GET4BIN ;GET FIRST NUMBER
SLL FIRST ;SAVE IT
CALL GET4BIN ;GET SECOND NUMBER
SLL SECOND ;SAVE IT
MOV B,H ;BC = SECOND NUMBER
MOV C,L
LHEL FIRST ;HL = FIRST NUMBER
DAD B ;HL = HL + BC
SLLD SUM ;SAVE SUM
LHEL FIRST ;HL = FIRST NUMBER
ORA A ;CLEAR CARRY
MOV A,L ;HL = HL - BC - CARRY
SUB C
MOV L,A
MOV A,H
SBB B
MOV H,A
PUSH H
POP  B ;BC = DIFFERENCE
LXI  R,HEXMSG2+7 ;CONVERT FOR PRINTING
CALL  CNVT16
LHLD SUM ;NOW PREPARE SUM FOR PRINTING
PUSH R
POP B ;BC = SUM
LXI R,HEXMSG1+6
CALL CNVT16
LXI D,HEXMSG1 ;PRINT SUM & DIFFERENCE
CALL PRINT
CALL CRLF
JMP MENU ;RETURN TO MENU

*** MTS COMMANDS - INITIATED BY HOST IN ALL CASES ***

* DWNLD - HEX FILE DOWNLOAD FROM DISK TO MDS MEMORY *

DWNLD LXI D,DWNLEMSG ;PRINT VERIFICATION MESSAGE
CALL PRINT
CALL GETFILENAME ;GET & CHECK FILENAME
LXI D,FCB ;OPEN FILE
CALL OPENFILE
CPI 255 ;FILE FOUND?
JNZ OPENOK ; YES
MVI A,13 ; NO, ERROR
JMP ERROR
OPENOK MVI A,`W' ;SEND DOWLOAD CMD TO MDS
CALL MSLCMD
XRA A ;RESET CONTINUATION &
STA  CONTFLG ; FIRST THROUGH LOOP FLAGS
STA FIRSTIME
RLFILE LXI R,DSKBUFF ;POINTER TO DISK BUFFER
CALL REAdSK ;READ IN AS MUCH AS POSSIBLE
LXI R,DSKBUFF ;NOW CONVERT IT TO BINARY &
   SEND IT TO MES
RECHL MOV A,`:' ;FIND `:' AS RECORD START
CPI JZ RECLen ;FOUND IT
INK R
CALL EOFCK ;END OF FILE/BUFFER?
JMP RECHL ; NO, TRY AGAIN
RECLen MVI B,0 ;INIT. CHECKSUM
CALL HEXBIN ;GET RECORD LENGTH
ORA A ; IF RECLen=0, THEN DONE
JZ DWNLDNE ; DONE
STA BUFFCNT ;SAVE THE RECLen
MOV C,A ; NOT DONE - SAVE RECLen
CALL METADATAOUT ;SEND IT TO MDS
CALL GETSADR ;GET START ADDRESS
LDA FIRSTIME ; IF FIRST TIME THROUGH LOOP
RRC            ; THEN SAVE ADDR FOR LATER
JC  RECLEN1    ; NOT FIRST TIME
LDR A          ; SET THE FLAG
STA FIRSTIME   ; AND SAVE THE ADDRESS
SLLD START     ; SAVE OTHER LOAD ADDRS
CALL ADDROUT   ; SEND ADDRESS TO MLS
XCHG           ; GET BUFFER POINTER BACK
CALL HXBIN     ; IGNORE RECORD TYPE
RECLEN1 SLL   ; SAVE OTHER LOAD ADDRS
CALL HEXBIN    ; GET DATA BYTE
CALL MDATAOUT  ; SEND DATA TO MDS
LDR C          ; INCREMENT RECORD LENGTH
JNZ HEXITDATA  ; MORE TO GET
INX H          ; GET NEXT RECORD
JMP RECED      ; GET STARTING LOAD ADDR
DWNLDNE LHLD START   ; PREPARE IT FOR PRINTING
PUSH H        ; GET STARTING LOAD ADDR
POP B         ; NOW READY THE FINISH ADDR
LXI H,DWNDONE1+20 ; GET RECLEN
CALL CNVT16
LHLE FINISH   ; GET RECLEN
LDA BUFFCNT
ADD L
MOV L,A
MOV A,H
ACI @
MOV H,A
PUSH H
POP B
LXI H,DWNDONE1+43
CALL CNVT16
LXI L,DWNDONE  ; PRINT COMPLETION MESSAGE
CALL PRINT
CALL DELAY
CALL HOSTTONE ; TELL MLS DONE
JMP MENU
GETSADR CALL HXBIN ; GET STARTING LOAD ADDRESS
MOV L,A ; FOR RECORD
CALL HXBIN
MOV B,A
XCHG           ; HL = LOAD ADDR
RET
CHECKIT CALL HXBIN ; CHECK FOR CORRECT CHECKSUM
XRA A          ; UL = BUFFER ADDRESS
ADD B          ; SC = BUFFER POINTER
RZ             ; SHOULD BE ZERO
OK

76
MVI A,14 ;CHECKSUM ERROR
JMP ERROR

* UPLD - HEX FILE UPLOAD (SAVE) OF MRS MEMORY TO DISK *

UPLD MVI A,128 ;INIT. BUFFER COUNT
STA BUFFCNT
LXI E,UPLOAD ;PRINT VERIFICATION MESSAGE
CALL PRINT
CALL GETFILENAME ;GET FILENAME & CHECK IT
LXI E,FBC
CALL DELETE ;DELETE ANY EXISTING FILE
CALL CREATE ;CREATE A NEW FILE
CPI 255 ;CREATE OK?
JNZ UPLD01 ;YES
MVI A,16 ;NO, OUT OF DIRECTORY SPACE
JMP ERROR
UPLD01 CALL BUFFRD ;GET ADDRESS INPUTS
MVI A,1 ;ONE DELIMITER ALLOWED
CALL SCAN ;SCAN OK
JNC UPLD1 ;SCAN ERROR
MVI A,2 ;ERROR
JMP ERROR
UPLD1 CALL GET4BIN ;GET MDF START & FINISH
SHLD START ;ADDRESSES FOR UPLOAD
CALL GET4BIN
SHLD FINISH
XCHG ;LE = FINISH ADDRESS
LHLD START ;CHECK FOR START > FINISH
MOV A,E
SUB L
MOV A,D
SBB H
JNC UPLE2 ;OK
MVI A,17 ;ERROR - START > FINISH
JMP ERROR
ULE12 MVI A,'U' ;SEND UPLOAD CMD TO MRS
CALL MDSCMD
LHLD START ;SEND START & END ADDRESSES
CALL ADDROUT
LXI H,ESKBUFF
CALL ADDROUT
UPLD3 MVI A,',' ;STORE RECORD HEADER
CALL BUFFCK
CALL WRITELN ;STORE RECORD LENGTH
CALL WRITADJR ;STORE STARTING LOAD ADDR
; & RECORD TYPE
CALL WRITTATA ;GET AND STORE DATA
CALL WRITCKS ;STORE CHECKSUM & CR,LF

77
JMP UPLD3 ;DO ANOTHER HEX RECORD
ORITLN01 XRA A
;WRITE LENGTH, ALTERNATE
AJMP WRITLEN1 ;ENTRY FOR ZERO RECLN
VRITLEN MVI A,16 ;ALL RECORDS HAVE RECLN=16
EXCEPT THE LAST
MVI B,0 ;INIT. CHECKSUM
CALL BINHEX ;CNVRT TC HEX ASCII & STORE
JMP WRITALLR LrA START+l ;STORE RECORD START ADDR
WRITALLR LDA START+1 CALL BINHEX
LDA START CALL BINHEX
PUSH H CALL BINHEX
SHL D CALL BINHEX
POP A CALL BINHEX
CALL BINHEX ;STORE RECORD TYPE
WRITDATA MVI C,16 ;DATA COUNTER
WRITDATA1 CALL MLSIN ;GET DATA FROM MLS
LDA MDSRDYF ;MORE DATA OR MLS LONE?
RRC JC WRITrNE ; MLS LONE
CALL BINHEX ;MORE DATA
DCR C ;16 BYTES YET?
RZ ; YES
JMP WRITDTAI ;NO, CONTINUE
WRITDNE XRA A DCR C ;FILL REMAINdER OF RECORE
JZ WRITDNE1 ; WITH ZEROS
CALL BINHEX
CALL WRITENE ;STORE CHECKSUM
CALL WRITEND ;STORE LAST RECORD
LLA BUFFCNT ;IS BUFFER FULL?
MOV A,128
CPI JZ WRITENE1 ;YES
WRITEN001 MOV M,EOF ;NO, FILL REMAINdER WITH INX ;EOF's
INX B ;LONE WITH FILL?
CPI JNZ WRITEN001 ;NO, CONTINUE
JP WRITEN001 ;YES, WRITE RECORD TO DISK
CALL CLOSFILE ;CLOSE THE FILE
CALL PRINT ;PRINT COMPLETION MESSAGE
CALL DELAY ;PRINT COMPLETION MESSAGE
CALL MENU
**EXAM - EXAMINE/SET MLS MEMORY LOCATION(S) **

**EXAM**
```
LXI E,EXAMS
```

**CALL**
```
PRINT
BUFFRD
XRA A
SCAN
EXAM01
ERROR
```

**JNC**
```
EXAM01
```

**MVI**
```
A,2
```

**JMP**
```
ERROR
```

**EXAM01**
```
GET4BIN
START
MVI A,'X'
MDSIN
ADDCNT
```

**CALL**
```
GET4BIN
SORT
MDSIN
```

**LHLD**
```
START
```

**MVI**
```
A,128
```

**STA**
```
BUFFCNT
```

**BUFFCK**
```
MOV M,A
```

**INX**
```
H
```

**LDA**
```
BUFFCNT
```

**ECR**
```
A
```

**JZ**
```
WRITEIT
```

**STA**
```
BUFFCNT
```

**RET**
```
```

**BUFFEND**
```
MVI A,A
```

**CALL**
```
BUFFCK
WRLNLN01
XCHG
MVT A,'X'
```

**LXI**
```
H,0000
```

**XLD**
```
H
```

**XCHG**
```
MVI A,LF
```

**CALL**
```
BUFFCK
```

**RET**
```
```

**WRTCKS**
```
MCV A,B
```

**CMA**
```
A
```

**INR**
```
CALL BINHEX
MVI A,CR
CALL BUFFCK
MVI A,LF
CALL BUFFCK
RET
```

**WRTEND**
```
MVI A,','
CALL BUFFCK
CALL WRITNLN01
XCHG
MVT A,'X'
CALL WRITAILR
CALL WRTCKS
RET
```

**BUFFCK**
```
MOV M,A
```

**INX**
```
H
```

**LDA**
```
BUFFCNT
```

**ECR**
```
A
```

**JZ**
```
WRITEIT
```

**STA**
```
BUFFCNT
```

**RET**
```
```

**WRTCKS**
```
A,B
```

**CMA**
```
A
```

**INR**
```
CALL BINHEX
MVI A,CR
CALL BUFFCK
MVI A,LF
CALL BUFFCK
RET
```

**WRTEND**
```
MVI A,','
CALL BUFFCK
CALL WRITNLN01
XCHG
MVT A,'X'
CALL WRITAILR
CALL WRTCKS
RET
```

**BUFFCK**
```
MOV M,A
```

**INX**
```
H
```

**LDA**
```
BUFFCNT
```

**ECR**
```
A
```

**JZ**
```
WRITEIT
```

**STA**
```
BUFFCNT
```

**RET**
```
```

**WRTCKS**
```
A,B
```

**CMA**
```
A
```

**INR**
```
CALL BINHEX
MVI A,CR
CALL BUFFCK
MVI A,LF
CALL BUFFCK
RET
```

**WRTEND**
```
MVI A,','
CALL BUFFCK
CALL WRITNLN01
XCHG
MVT A,'X'
CALL WRITAILR
CALL WRTCKS
RET
```

**BUFFCK**
```
MOV M,A
```

**INX**
```
H
```

**LDA**
```
BUFFCNT
```

**ECR**
```
A
```

**JZ**
```
WRITEIT
```

**STA**
```
BUFFCNT
```

**RET**
```
```

**WRTCKS**
```
A,B
```

**CMA**
```
A
```

**INR**
```
CALL BINHEX
MVI A,CR
CALL BUFFCK
MVI A,LF
CALL BUFFCK
RET
```

**WRTEND**
```
MVI A,','
CALL BUFFCK
CALL WRITNLN01
XCHG
MVT A,'X'
CALL WRITAILR
CALL WRTCKS
RET
```

**BUFFCK**
```
MOV M,A
```

**INX**
```
H
```

**LDA**
```
BUFFCNT
```

**ECR**
```
A
```

**JZ**
```
WRITEIT
```

**STA**
```
BUFFCNT
```

**RET**
```
```

**WRTCKS**
```
A,B
```

**CMA**
```
A
```

**INR**
```
CALL BINHEX
MVI A,CR
CALL BUFFCK
MVI A,LF
CALL BUFFCK
RET
```

**WRTEND**
```
MVI A,','
CALL BUFFCK
CALL WRITNLN01
XCHG
MVT A,'X'
CALL WRITAILR
CALL WRTCKS
RET
```

**BUFFCK**
```
MOV M,A
```

**INX**
```
H
```

**LDA**
```
BUFFCNT
```

**ECR**
```
A
```

**JZ**
```
WRITEIT
```

**STA**
```
BUFFCNT
```

**RET**
```
```

**WRTCKS**
```
A,B
```

**CMA**
```
A
```

**INR**
```
CALL BINHEX
MVI A,CR
CALL BUFFCK
MVI A,LF
CALL BUFFCK
RET
```

**WRTEND**
```
MVI A,','
CALL BUFFCK
CALL WRITNLN01
XCHG
MVT A,'X'
CALL WRITAILR
CALL WRTCKS
RET
```

**BUFFCK**
```
MOV M,A
```

**INX**
```
H
```

**LDA**
```
BUFFCNT
```

**ECR**
```
A
```

**JZ**
```
WRITEIT
```

**STA**
```
BUFFCNT
```

**RET**
```
```

**WRTCKS**
```
A,B
```

**CMA**
```
A
```

**INR**
```
CALL BINHEX
MVI A,CR
CALL BUFFCK
MVI A,LF
CALL BUFFCK
RET
```

**WRTEND**
```
MVI A,','
CALL BUFFCK
CALL WRITNLN01
XCHG
MVT A,'X'
CALL WRITAILR
CALL WRTCKS
RET
```

**BUFFCK**
```
MOV M,A
```

**INX**
```
H
```
PUSH H ;SAVE ADDR. BEING EXAMINEE
MOV C,A ;C = MDSDATA
LXI H,EXAMSG2+1 ;CONVERT DATA FOR PRINTING
CALL CNVT8
POP B ;GET ADDR. BACK,
PUSH B ; BUT SAVE IT
LXI H,EXAMSG1 ;CONVERT ADDR. FOR PRINTING
CALL CNVT16
XCHG ;LE = EXAMSG1
CALL PRINT ;PRINT MDS ADDR. & DATA
CALL BUFFRD1 ;GET REPLACEMENT DATA
ORA A ; IF NO INPUT, THEN PUT OLD
JZ NOSET ; DATA BACK
XRA A ;NO DELIMITERS ALLOWED
CALL SCAN ;SCAN OK
JNC EXAM02 ;SCAN OK
MVI A,2 ;INPUT ERROR
JMP ERROR ;START OPTION OVER
EXAM02 CALL CKPERIOD ;IF INPUT WAS A PERIOD,
ORA A ; THEN DONE
JZ EXAM2 ;NO PERIOD, GET DATA
JNC EXDONE ;YES - ALL DONE
JZ EXAM2 ;NO PERIOD GET DATA
MC A,3 ;PERIOD ONLY?
JMP ERROR ;NO - PERIOD + DATA IS
MVI A,2 ;ILLEGAL, START OVER
JMP ERROR
EXAM2 CALL GET2BIN ;SEND NEW DATA
MOV A,L
JMP SET1
NOSET LEA MDSDATA ;GET OLD DATA
SET1 CALL MDATAOUT
POP H ;BUMP ADDRESS FOR EXAM/SET
INX H
JMP EXAM1 ;GET MORE DATA FROM MDS
EXDONE CALL HOSTDONE ;SIGNAL MDS DONE
JMP MENU ;BACK TO MENU

* FILL - FILL MDS MEMORY LOCATION(S) WITH SPECIFIED DATA *

FILL LXI H, FILLMSG ;PRINT VERIFICATION MESSAGE
CALL PRINT
CALL BUFFRD ;GET INPUT ADDRESSES + FILL ;DATA
MVI A,2 ;TWO DELIMITERS REQUIRED
CALL SCAN ;CHECK FOR THEM
JNC FILL1 ;SCAN OK
MVI A,2 ;JMP ERROR
JMP ERROR ;START OPTION OVER
FILL1 CALL GET4BIN ;GET START ADDRESS
SHL START ;SAVE IT
CALL GET4BIN ;GET FINISH ADDRESS
SHL FINISH ;SAVE IT TOO
CALL GET2BIN ;GET FILL DATA
MOV A, L ;A = DATA
STA CONSATA ;SAVE IT
MVI A, 'F' ;SEND FILL CMD TO MES
CALL MDSCMD
LHLL START ;SEND START ADDR. TO MES
CALL ALEROUT
LHLD FINISH ;SEND FINISH ADDR. TO MES
CALL ALEROUT
LEA CONSATA ;SEND FILL DATA TO MES
CALL MDATAOUT
MVI A, 1
STA SYSTAT
CALL ALEROUT
LLA CONSATA ;SEND FILL DATA TO MES
CALL MDATAOUT
MVI A, OFFH ;MES IN CONTROL
STA SYSSTAT
STA MDSRDYF
JMP MENU ;RETURN TO MENU

* SEND 16 BIT ADDRESS TO MES - CALL WITH HL = ADDRESS *

ALEROUT MOV A, H ;MSB FIRST
CALL MLAACUT
MOV A, L ;THEN LSB
CALL MLAACUT
RET ;BACK TO CALLER

= CSET - CONTINUOUS SET MDS MEMORY WITHOUT EXAMINE =

CSET LXI E, CSETMSG ;PRINT VERIFICATION MESSAGE
CALL PRINT
MVI A, 0FFH ;INIT. CONSOLE READ BUFFER
STA CONBUFF ;TO 255 CHARACTERS MAX
CALL BUFFRD ;GET START ADDRESS
XRA A ;NO DELIMITERS ALLOWED
CALL SCAN
JNC CSET01 ;SCAN OK
MVI A, 2 ;INPUT ERROR
JMP ERROR ;START OPTION OVER

CSET01 CALL GET4BIN
SHL START ;SEND CSET CME TO MES
MVI A, 'C' ;SEND CSET CMD TO MES
CALL MDSCMD
LHLL START ;SEND START ADDR. TO MES
CALL ALEROUT
JMP CSET11
CSET1 CALL CRLF ;GET REPLACEMENT DATA TILL
CSET11 CALL BUFFRD ;BUFFER FULL OR <CR>

81
CALL SCAN               ;LOOK FOR ESCAPE
CALL CKPERIOD           ;CHECK FOR PERIOD IN INPUT
ORA A                   ; PERIOD, GET DATA
JZ CSET2                ; NO PERIOD, GET DATA
RAR                      
JC CSET3                ; PERIOD ONLY - ALL DONE
MVI A,3                  ; INPUT ERROR,
JMP ERROR                ; START OPTION OVER
CSET2 CALL STAR         ; PRINT A LEADING STAR
                ; PRIOR TO VALIDATION DATA
CSET21 CALL GET2BIN     ; GET DATA
MOV A,L                  ; SEND IT TO MS3
CALL MI_DATAOUT         ; PRIOR TO VALIDATION DATA
MOV C,A                  ; ALL DONE
LXI H,DATAMSG+1          ; SEND IT TO CONSOLE FOR
CALL CNVTS               ; VERIFICATION
XCHG                      ; LE = ADR. OF DATAMSG
CALL PRINT               ; LE = CURRENT CONUFF PTR
MOV A, B                 ; AT END OF BUFFER?
CPI OFFH                 ; YES, START OVER
JZ CSET1                 ; YES, START OVER
CPI 00                    ; NO, GET MORE DATA
JMP CSET21               ; NO DATA TO SEND, SIGNAL
CSET3 CALL HOSTDONE     ; MDS DONE
                ; RETURN TO MENU
JMP MENU

* EXEC - EXECUTE MIS MEMORY FROM A SPECIFIED ADDRESS *

EXEC LXI L, EXECMSG     ; PRINT VERIFICATION MESSAGE
CALL PRINT               ; GET START ADDRESS
XRA A                    ; NO DELIMITERS ALLOWED
CALL SCAN                ; SCAN OK
JNC EXEC1                ; NO DELIMITERS ALLOWED
MVI A, 2                 ; ERROR
JMP EXEC1                ; START OPTION OVER
EXEC1 CALL GET4BIN       ; GET START ADDRESS
SHL START                ; SAVE IT
LXI L, EXMSG             ; SEE IF DATA FROM MDS TO
CALL PRINT               ; CONSOLE OR NCT
EXEC11 CALL CONSTAT      ; WAIT FOR RESPONSE
RRC                      
JNC EXEC11               ; LOOP
CALL CONIN               ; GET RESPONSE
ORI 20H                  ; FORCE TO LOWER CASE
CPI 'y'                  ; CONSOLE INPUT FROM MDS?
JNZ EXEC2                ; NO, SEND CM & RETURN TO
                ; MENU
LXI D,EXMSG2  ; GIVE ESCAPE METHODS
CALL PRINT
MVI A, ´E´  ; YES, SEND CYL TO MDS &
CALL MSCMD  ; LOOP WAITING FOR DATA
LHLL START  ; OR DONE FROM MDS OR ESC
 FROM CONSOLE
CALL ADRROUT
EXE022 CALL MILSIN  ; LOOP AT MILSIN TILL ESC
 FOR 'Q' OR DATA
MOV E,A  ; SAVE DATA FROM MDS
LEA M'SRELFYF  ; SEE IF M'S WANTS INPUT
ORA A
JNZ GETINP  ; YES
CALL CONSOUT  ; NO, SEND IT TO CONSOLE
JMP EXEC20  ; WAIT FOR MORE
GETINP CALL CONSOUT  ; GET INPUT FROM KEYBOARD
RRC
JNC GETINP
CALL CONSIN
CALL 'MTAOU'T  ; SEND IT TO M'S
XRA A  ; RESET M'S RLYF
STA M'SRLYF
JMP EXEC2  ; LOOP AGAIN
EXEC2 MVI A, ´E´  ; SEND M'S EXEC CMD
CALL MDSCMD
LHLL START  ; SEND START ADDRESS TO M'S
CALL ADRROUT
MVI A,1  ; MORE THAN ONE DELIMITER?
STA SYSSAT
JMP MENU  ; SET M'S CONTROL FLAG
JMP MENU  ; BACK TO MENU

* LOCATE - LOCATE A SPECIFIED BYTE SEQUENCE IN MDS MEMORY *

LOCATE CALL CLRBUFF  ; CLEAR REAL BUFFER
LXI D,LOCMSG  ; PRINT VERIFICATION MESSAGE
CALL PRINT
MVI A, ´OFF´  ; INIT. CONSOLE READ BUFFER
STA CONBUFF  ; TO 255 CHARACTERS MAX
CALL BUFFRL  ; GET ADDRESS(ES)
XRA A  ; ANY DELIMITERS ?
CALL SCAN
JNC LOCATE1  ; NO, USE DEFAULT FINISH
 CALL ADDRESS
MVI A,1  ; MORE THAN ONE DELIMITER?
CALL SCAN
JNC LOCATE1  ; NO, GET OPTIONAL FINISH
 CALL ADDRESS
MVI A,2  ; MORE THAN 2 DELIMITERS
JMP ERROR  ; ERROR, START OPTION OVER
LOCATE1 CALL GET4BIN  ; GET START ADDRESS
SHLE START
JMP LCC1 ; NO COMMA, FINISH ADDRESS
; DEFAULTS TO OFFFFH —
; GET DATA
LOC21 CALL GET4BIN ; GET START ADDRESS
SHLD START
CALL GET4BIN
SHLD FINISH
JMP LOCDATA
LOC1 LXI h,FFFFFH ; SAVE DEFAULT FINISH ADDRESS
SHLD FINISH
LOCDATA MVI A, 'L' ; SEND LOCATE CMD TO MDS
CALL MISCMD
LHLD START ; SEND START ADDRESS TO MDS
CALL ADDROUT
LHLD FINISH ; SEND FINISH ADDRESS TO MDS
CALL ADDROUT
MVI A, 16 ; 16 BYTES MAX
PUSH PSW ; SAVE BYTE COUNT
LCLIATA1 CALL BUFFREC ; GET SEARCH SEQUENCE
CALL SCAN ; LOOK FOR ESCAPE
CALL STAR ; PRINT A STAR
LOCDATA2 CALL GET2BIN ; GET A BYTE
MOV A, L
CALL MDATAOUT ; SEND IT TO MDS
MOV C, A
LXI h, DATAMSG + 1 ; & TO CONSOLE FOR
CALL CVT8 ; VERIFICATION
XCHG
CALL PRINT
XCHG
MOV A, B ; AT END OF BUFFER?
CPI OFFH
JZ LOC5 ; YES, WAIT FOR SEARCH
CPI 00
JZ LOC5 ; YES, WAIT FOR SEARCH
POP PSW ; NO, GET BYTE COUNT
DCR A ; 16 BYTES YET?
PUSH PSW ; SAVE BYTE COUNT
JZ LOC5 ; YES, WAIT FOR SEARCH
JMP LOCDATA2 ; NOT AT END OR 16 BYTES
LOC5 CALL HOSTRDY ; TELL MDS TO SEARCH
CALL MINSIN ; GET MDS RESPONSE
ORI 80H ; LOOKING FOR ASCII
CPI 'F' ; BYTE SEQ. FOUND?
JZ FOUND ; YES
LXI D, NOTFOUND ; PRINT NOT FOUND MESSAGE
JMP ERROUT ; BACK TO MENU
FOUND LXI D, FOUNDEMSG ; PRINT FOUND MESSAGE
CALL PRINT
CALL MDSIN ; GET FOUND ADDRESS MSB
MOV B, A
CALL MDSIN ; GET LSB OF ADDR
MOV C, A
LXI H, FOUNDMS1 ; CONVERT ADDR. FOR PRINTING
XCHG
CALL PRINT ; PRINT ADDRESS
JMP MENU ; BACK TO MENU

* DUMP - DUMP MDS MEMORY LOCATION(S) *

LUMP CALL CLRBUFF ; CLEAR REAL BUFFER
LXI D, DUMPMSG ; PRINT VERIFICATION MESSAGE
CALL PRINT
CALL BUFFREP ; GET ADDRESS(ES)
XRA A ; ANY DELIMITERS?
CALL SCAN
JNC LUMP01 ; NO
MVI A, 1 ; MORE THEN ONE DELIMITER?
CALL SCAN
JNC LUMP010 ; NO, GET OPTIONAL FINISH ADDRESS
MVI A, 2 ; MORE THAN ONE DELIMITER
JMP ERROR ; ERROR, START OPTION OVER
LUMP01 CALL GET4BIN ; GET START ADDRESS
SHLD START
JMP LUMP1 ; NO COMMA
LUMP010 CALL GET4BIN ; GET START ADDRESS
SHLD START
SHLD FINISH
JMP DUMP2
LUMP1 LHLD START ; MAKE FINISH ADDRESS =
LXI B, 0100H
DAD B
SHLD FINISH
LUMP2 MVI A, 'D' ; SEND LUMP CMD TO MDS
CALL MDSCMD
LHLD START ; SEND START ADDRESS TO MDS
CALL ADDROUT
LHLD FINISH ; SEND FINISH ADDRESS TO MDS
CALL ADDROUT
LUMP3 LXI D, DUMPMSG3 ; ASCII DATA STORAGE
PUSH D
CALL MSG3INIT ; INIT. ASCII STORAGE
CALL MDSIN ; GET BYTE
MOV C, A
LDA MISRLYF ; MIS DONE TRANSmitING DATA?
ORA A
JNZ DUMPDONE ; YES
MOV A,C
STA MISEDATA ; NO - SAVE DATA
LHLD START ; BC = START ADDRESS
MOV B,H
MOV C,L
LXI H,DUMPMSG1
CALL CNVT16
XCHG CALL PRINT ; SIXTEEN BYTES PER LINE
MVI B,16 ; RECALL ASCII DATA STORAGE
LEA MISEDATA ; LOCATION
MOV C,A ; IS DATA ASCII PRINTABLE?
CPI 20H
JNC EMPETA1 ; YES
CALL SPERIOI ; NO - STORE A PERIOD
JMP DMPDTA2
LMPDTA1 CPI 80H ; GREATER THEN ASCII
CNC SPERIOI ; YES, STORE A PERIOD
LMPDTA2 STAK D ; STORE DATA AS IS
MOV A,C ; RESTORE ORIGINAL DATA
INX D ; BUMP STORAGE ADDRESS
PUSH D ; AND SAVE IT
LXI E,DUMPMSG2+1 ; NOW CONVERT DATA TO HEX
; AND PRINT IT
PUSH B ; SAVE COUNT
CALL CNVT8
XCHG CALL PRINT
POP B ; GET COUNT BACK
LCR B ; 16 BYTES YET?
JZ NXTLINE ; YES
CALL MISIN ; NO - GET NEXT BYTE
STA MISEDATA ; SAVE NEW DATA
MOV C,A ; MSD DONE TRANSMITING DATA?
LEA MISREYF
ORA A
JNZ NXTLINE ; YES
JMP DUMPDATA ; NO - GET NEXT LINE OF DATA
SPERIOI MVI A,.'.' ; STORE A PERIOD IF NOT A
RET ; PRINTABLE ASCII CHAR.
NXTLINE LXI E,DUMPMSG3 ; PRINT ASCII CHARACTERS
CALL PRINT
LXI B,0012H ; GO TO NEXT LINE
LHLD START ; BUMP NEW LINE START ADDRESS
LAL B ; BY SIXTEEN BYTES
SHLD START ; SAVE IT
POP E ; GET GARBAGE OFF STACK
LDA *MISREYF ; LONE?
JNZ DUMPEONE ; YES
CALL CRLE ; START NEW LINE
LUMPONE L A ; DUMP TILL DONE
S... MESRLYF
CALL CRLE
CALL CRLE ; START NEW LINE
CALL CRLE ; CLEAR MSL DUMP TRANSMITTING FLG
CALL INPUT Lמשינש ; ANOTHER DUMP?
JNC DUMPEONE ; SCAN OK
MVI A,2 ; ERROR
JMP ERROR ; START OPTION OVER
LUMPONE INX L ; POINT TO END OF BUFFER
DCL B ; THERE YET?
JNZ DUMPONE ; THERE YET?
LAX L ; NO, LOOP
ORI 20H ; CONVERT TO LOWER CASE
CPL A "a"
JZ LUMPAGAIN ; YES - LUMP AGAIN FROM PREVIOUS FINISH ADDRESS
JMP MENU ; NO - RETURN TO MENU
LUMPAGAIN INX H ; MAKE FINISH+1 = NEW START ADDRESS
SHLD START
JMP LUMP256 ; LUMP 256 MORE BYTES
MSG3INIT MVI B,17 ; INIT. ASCII DATA STORAGE
LXI D, DUMPMS33 ; AREA TO ALL $'S
MVI A, "$"
MSG31 STAX L
DCR B
JMP MSG31
RCNT2HST MVI B,255 ; CLEAR CONSOLE READ BUFFER
LXI D, CONBUFF+1
MVI A, 00 ; PUT IN ALL ZEROS
JMP MSG31

* RCNT2HST - RETURN CONTROL TO HOST *

RCNT2HST LDA SYSSTAT ; GET SYSTEM STATUS
ORA A
JZ MENU ; HOST ALREADY IN CONTROL
MVI A, "Q" ; SEND ESCAPE TO MD5
CALL MSGMCMD
XRA A
STA SYSSTAT
LXI L, ABORTEIM ; PRINT MLS ABORTED VERIFICATION
CALL PRINT
CALL DELAY
JMP MENU
*** UTILITY SUBROUTINES ***

* PRINT A STRING TO THE CONSOLE
* CALL WITH LE = STARTING ADDRESS OF STRING *

PRINT   PUSH    PSW      ;SAVE EVERYTHING
         PUSH    B
         PUSH    L
         PUSH    H
         MVI    C,PRTSTRG ;OUTPUT STRING TO CONSOL
         CALL    BIOS
         POP    H      ;RESTORE ALL REGISTERS
         POP    D
         POP    B
         POP    PSW  
         RET      ;BACK TO CALLER

* STATSYS - DISPLAY SYSTEM STATUS *

STATSYS CALL    CRLF
             CALL    CRLF
             LDA    SYSSTAT     ;GET SYSTEM STATUS FLAG
             ORA    A
             LXI    L,SYSMSG+15 ;HOST IN CONTROL
             JZ    SYS1
             LXI    H,MDSMSG    ;MDS IN CONTROL
             JMP    SYS1+3
             LXI    H,HOSTMSG  ;PUT 'HOST' IN MESSAGE
             CALL    MOVESTR
             LDA    MENUSUPPF ;GET MENU SUPPRESSION FLAG
             ORA    A
             LXI    D,SYSMSG+33 ;NO SUPPRESSION
             JZ    SYS3
             LXI    H,YESMENMSG ;SUPPRESSION
             JMP    SYS3+3
             LXI    H,NOMENMSG
             CALL    MOVESTR
             LXI    D,SYSMSG ;PRINT SYSTEM STATUS
             CALL    PRINT
             CALL    MENPMPT ;PRINT MENU PROMPT
             RET      ;RETURN TO CALLER

MOVESTR    MOV    A,M   ;HL = STRING TO MOVE
             CPI    " " ;IE = DESTINATION ADDRESS
             RZ     :RETURN IF MOVE DONE
             STAX   L   ;NOT DONE
             INX    L
             INX    H
             JMP    MOVESTR ;MOVE NEXT CHARACTER

* MENPMPT - PRINT MENU PROMPT *
** ROUTINES TO GET AND CHECK FILENAMES FOR VALIDITY **
** ONLY INTEL 'HEX' FILES ARE SUPPORTED BY THIS VERSION **

* GETFILEN - INITIATE CALLS FOR INPUTTING FILENAME AND
* MAKING APPROPRIATE CHECKS *

GETFILEN CALL CLRBUFF ;CLEAR CONSOLE INPUT BUFFER
LXI L,FILNAME ;PROMPT FOR FILENAME
CALL PRINT
CALL BUFFRD ;GET FILENAME
CALL FILENCK ;TO CHECKS ON FILENAME
ORA A ;SEE IF ANY ERRORS
JZ GETFNI ;NO ERRORS
CALL ERROR ;ERRORS
JMP GETFILEN ;START OVER

GETFNI CALL MOVFN ;MOVE FILENAME TO PCB
CALL UCASE ;CONVERT ALL FILENAME
RET ; ALPHABETICS TO UPPER CASE

* FILENCK - INITIATE ALL FILENAME CHECKS
* RETURN A = 00 IF NO ERRORS
* = ERROR NUMBER IF ERRORS IN FILENAME *

FILENCK CALL SCANQ ;SCAN FILENAME FOR '?'
RRC
JNC FNCK1 ;NONE FOUND
MVI A,7 ;ERROR - NO AMBIGUOUS
RET ; FILENAMES
FNCK1 CALL SCANCOL ;CHECK FOR ':' AND PROPER
RRC
JNC FNCK2 ;SCAN OK
MVI A,8 ;TOO MANY COLONS
RET
FNCK2 CALL SCANUM ;CHECK FOR TOO MANY OR TOO
RRC
JNC FNCK3 ;NO ERROR
MVI A,9 ;ERROR
RET
FNCK3 CALL CKPERIOD ;CHECK FILENAME INPUT FOR
ORA A ; A PERIOD
JZ FNCK4 ;NONE FOUND
CALL SCANHEX ;ONE PERIOD, CHECK FCR
RRC ;'HEX' FILETYPE
JNC FNCK4 ;FILETYPE OK
MVI A,10 ;ONLY 'HEX' FILETYPES ARE
RET ;SUPPORTED

FNCK4 XRA A ;CHECK FOR ESCAPE ANR
CALL SCAN ;OTHER DELIMITER ERRORS
JNC FNCK5 ;NONE FOUND
MVI A,11 ;NO SPACES ALLOWED IN
RET ;FILENAME

FNCK5 CALL SCANINV ;CHECK FOR NON-PRINTABLE
RRC ;CHARACTERS IN FILENAME
JNC FNCK6 ;NONE FOUND
MVI A,12 ;ERROR
RET ;FILENAME OK

FNCK6 XRA A ;NO ERRORS DETECTED
RET ;FILENAME OK

* SCAN - SCAN FILENAME FOR QUESTION MARKS INDICATING AN
* AMBIGUOUS FILENAME
* RETURN A = 00 IF NONE FOUND
* = OFFH IF FOUND *

SCANQ PUSH B
PUSH C
PUSH H
XCHG ;HL = BUFFER + 1
MOVC,M
;GET BUFFR COUNT

SCANQ01 INX H
MOV A,M
;LOOK FOR '?'
CPI
JZ SCANQ1 ;FOUND ONE
DCR C ;KEEP LOOKING?
JNZ SCANQ01 ;SCAN NOT DONE
XRA A ;SCAN DONE - NO ERRORS
JMP SCANQ1+2

SCANQ1 MVI A,OFFH ;AT LEAST ONE '?' FOUND
POP H
POP D
POP B
RET

* SCANCOL - SCAN FILENAME FOR A ': ' THEN LOOK FOR PROPER
* DRIVE SELECT CODE (ONLY CURRENT DRIVE IN USE
* IS SUPPORTED, OTHERS ARE IGNORED)
* - A ': ' IN ANY OTHER POSITION IN THE FILENAME IS
* NOT LEGAL
* RETURN A = 00 IF NO ERROR
* = OFFH IF AN ILLEGAL ': ' IS FOUND *

SCANCOL PUSH B
PUSH E
PUSH H
CALL CURRISK
ORI A, 40H
INR A
STA CURRENT
XCHG
MOV C, M
INX H
INX H
DCR C
MOV A, M
CPI
JNZ SCANCOL1
DCX H

; THE ONLY ':' WOULD BE HERE

DCR C
MOV CPI
JNZ SCANCOL2
INX H
DCR C

; CONTINUE SCAN

SCANCOL1 DCX H
INR C
SCANCOL1 MOV A, M
CPI
JNZ SCANCOL3
DCR C
JNZ SCANCOL2
INX H
DCR C

; SCAN NOT DONE

SCANCOL2 INX H
MOV A, M
CPI
JNZ SCANCOL3
DCR C
JNZ SCANCOL2

; SCAN NOT DONE

SCANCOL3 MOV A, OFFH
POP H
POP E
POP B
RET

91
* SCANHEX - SCAN FILETYPE FOR 'HEX'
* RETURN A = 00 IF FOUND
* = OFFH IF NOT FOUND *

SCANHEX
PUSH B
PUSH D
PUSH H
XCHG
MOV C, M
;GET BUFFER COUNT

SCANHX1
INX H
MOV A, M
CPI PERIOD
JZ COMPARE
;FOUND IT

DCR C
JNZ SCANHX1
;KEEP LOOKING
JMP SCNHXER
;ERROR, NO PERIOD

COMPARE
INX H
MOV A, M
ANI OFEH
;FORCE TO UPPER CASE
CPI "H"
JNZ SCNHXER
;ERROR

MOV A, M
ANI OFEH
CPI "E"
JNZ SCNHXER
;ERROR

MOV A, M
ANI OFEH
CPI "X"
JNZ SCNHXER
;NO ERROR

XRA A
JMP SCNHXER+2
;ERROR

SCNHXER
MVI A, OFFH
POP H
POP D
POP B
RET

* SCANUM - SCAN FILENAME FOR TOO MANY OR TOO FEW CHARACTERS
* FILENAME IS CHECKED ONLY (8 CHARACTERS MAX,
* 1 CHARACTER MINIMUM)
* RETURN A = 00 IF NO ERROR
* = OFFH IF ERROR *

SCANUM
PUSH B
PUSH D
PUSH H
XCHG
MOV C, M
MVI B, 0
;E = # OF CHARACTERS IN FN

92
SCANUM1 INX H
    MOV A,M ; START COUNT AT `':`?
    CPI A
    JNZ SCANUM2 ; YES
    DCR B ; NC, START AT BEGINNING
    ECR C ; LONE YET?
    JZ SCANUM4 ; YES
    JMP SCANUM1 ; NO
SCANUM2 CPI PERIOD ; GO TO PERIOD OR BUFFER END
    JZ SCANUM4 ; PERIOD, LONE
    INR B ; KEEP COUNTING
    ECR C
    JZ SCANUM4 ; LONE
    JMP SCANUM1 ; LOOP
SCANUM4 XRA A ; < 1 CHARACTER?
    CMP B
    JZ SCANUM5 ; YES, ERROR
    MVI A,B ; > 8 CHARACTERS?
    CMP B
    JC SCANUM5 ; YES, ERROR
    XRA A ; NO ERRORS
    JMP SCANUM5+2
SCANUM5 MVI A,OFFH ; ERROR
    POP H
    POP C
    POP B
    RET

* SCANINV - SCAN FILENAME FOR NON-PRINTABLE CHARACTERS
* RETURN A = 00 IF NONE FOUND
*    = OFFH IF ANY FOUND *

SCANINV PUSH B
    PUSH D
    PUSH H
    XCHG MOV C,M
    MOV A,M
    CPI 20H ; < SPACE?
    JC SCANIN2 ; YES, ERROR
    DCR C ; DONE WITH SCAN?
    JNZ SCANINV1 ; NO
    XRA A ; YES, NO ERRORS
    JMP SCANIN2+2
SCANIN2 MVI A,OFFH ; ERROR
    POP H
    POP C
    POP B
    RET
* MOVFN - MOVE FILENAME FROM CONSOLE BUFFER TO FCB *

MOVFN
CALL PURGFCB ; PURGE AND SET UP FCB
LXI H,CONBUFF+1 ; GET BUFFER COUNT
MOV C,M
XCHG
INX D
INX D ; SEE IF IT'S A COLON
DEC C
LEAX D
CPI ":
JZ MOVIT01 ; YES
DCX E ; NO
INC C
JMP MOVIT ; START AT BUFFER START

MOVIT01
INX D ; START FROM COLON
DEC C
MOVIT LXI H,FCE+1
MOVI L,D
; MOVE THE FILENAME
CPI PERIOD ; UNTIL PERIOD OR END
; OF BUFFER
ZR ; DONE
MOV M,A ; STORE CHAR. IN FCB
INX H
INX E
DEC C ; AT END OF BUFFER?
ZR ; YES, MOVE DONE
JMP MOVIT1 ; NO, LOOP

* PURGFCB - PURGE FILE CONTROL BLOCK (FCB) AND SET IT UP *
* FOR ACCEPTING A FILENAME OF TYPE HEX *

PURGFCB
LXI H,FCE
LXI L,FCBMSG
MVI C,16 ; SET UP FIRST 16 BYTES

PURG01
LDA D
MOV M,A
DEC C ; 16 BYTES DONE YET?
JZ PURG1 ; YES
INX H
INX E
JMP PURG01 ; NO, LOOP

PURG1
LXI L,FCB+32 ; INITIALIZE CURRENT RECORD
XRA A ; BYTE IN FCR
STAX D
RET

* UCASE - CONVERT ALL FILENAME ALPHABETICS TO UPPER CASE *

UCASE
MVI C,8 ; 8 CHARACTERS MAX
LXI H,FCE+1
UCASE01
MOV A, M
CPI 7EH ; IS IT > LOWERCASE z?
JNC UCASE01 ; YES, OK
CPI 'a'
JC UCASE01 ; NO, IS IT < LOWERCASE a?
ANI $0FH ; MUST BE LOWER CASE
MOV M, A ; CONVERT IT TO UPPER CASE
UCASE01
INX H
LCR C ; ONE?
RZ
JMP UCASE01 ; YES, OK
JC UCASE01 ; NO, LOOP

* HEXBIN - CONVERT TWO HEX ASCII CHARACTERS TO ONE EIGHT
* BIT BINARY NUMBER
* - ALSO ADD IT TO CURRENT CHECKSUM IN E
* CALL WITH HL POINTING TO FIRST CHARACTER
* RETURN BINARY NUMBER IN A *

HEXBIN
INX E
MOV A, M ; GET FIRST DIGIT
CALL EOFCK ; END OF BUFFER/FILE?
CALL ASCHEX ; CONVERT TO PURE HEX
RLC
RLC
RLC
MOV E, A ; SAVE IT
INX E ; GET SECOND DIGIT
MOV A, M
CALL EOFCK
CALL ASCHEX ; CONVERT IT
ADD E ; COMBINE THEM
MOV E, A ; SAVE IT
ADD B ; ADD TO CHECKSUM
MOV B, A ; SAVE IT
MOV A, E ; GET BINARY NUMBER
RET

* ASCHEX - CONVERT HEX ASCII DIGIT TO PURE HEX DIGIT *

ASCHEX SUI '0'
CPI 10
RC
SUI 7
RET

* EOFCK - CHECK FOR END OF BUFFER/FILE *
* - IF END OF FILE THEN DOWNLOAD IS DONE
* - IF END OF BUFFER, READ MORE DISK & RETURN WITH
* THE FIRST CHARACTER IN A
* - OTHERWISE, RETURN WITH NO ACTION *

95
**BINHEX** – CONVERT AN EIGHT-BIT BINARY NUMBER TO TWO HEX ASCII CHARACTERS

- STORE THE CHARACTERS IN MEMORY POINTED TO BY HL

**CALL** WITH BINARY NUMBER IN A AND HL AS ABOVE *

```
BINHEX
PUSH PSW ;SAVE DATA
ALL B ;ALL TO CHECKSUM
MOV B,A ;SAVE IT
POP PSW ;GET DATA
MOV E,A ;SAVE IT IN E
ANI $0F0H ;PUT 4 MSB'S INTO LSB'S
RRC
RRC
RRC
CALL HEXASC ;CONVERT TO HEX ASCII
CALL BUFFCK ;STORE IT
MOV A,E ;GET DATA
ANI $0Fh ;NOW CONVERT LSB'S
CALL HEXASC
CALL BUFFCK ;STORE IT
RET
```

**HEXASC** – CONVERT A BINARY NUMBER TO A HEX ASCII CHAR.

```
HEXASC
CPI $0Ah
JC NUMBER ;IT IS 0–9
ADI 7 ;IT IS A–F
NUMBER ADI $30H ;ALL ASCII BIAS
RET
```

**DISK I/O ROUTINES**

**ALL ERROR CODES RETURNED ARE IN ACCORDANCE WITH CP/M AND MP/M CONVENTIONS**

**READSK** – READ THIRTY-TWO (32) 128-BYTE RECORDS FROM DISK

SET FLAG TO INDICATE IF ONLY A PARTIAL READ *
READSK  PUSH     B          ;SAVE B
       MVI       B,32       ;READ 32 RECORDS MAX
       CALL      DMASET    ;SET DMA ADDRESS
       CALL      READREC   ;READ A SINGLE RECORD
       CPI       0          ;GOOD READ?
       JZ        READMORE  ;YES, DO IT AGAIN
       CPI       1          ;EOF?
       JZ        READNE    ;YES, DONE
       MVI       A,15       ;NO, READ ERROR
       READMORE  ECR B     ;4K WORTH YET?
       JNZ       READSK1   ;NO, READ MORE
       MVI       M,EOF      ;YES, STORE END OF BUFFER
                 ;INDICATOR
       MVI       A,OFFH     ;SET CONTINUATION FLAG
       STA       CONTFLG   ;SET CONTINUATION FLAG
       POP       B          ;RESTORE B
       RET
       READNE  XRA A       ;RESET CONTINUATION FLAG
       STA       CONTFLG   ;POINT TO END OF LAST RECORD
       PUSH      B          ;ENSURE EOF MARKER IN BUFFER
       LXI       B,-128     ;RESTORE ORIGINAL B
       POP       B
       MVI       A,18       ;NO, OUT OF DISK SPACE
       CALL      ERROR      ;YES, DONE
       CALL      CLOSFILE   ;CLOSE THE FILE BUT
       CALL      DELETE     ;DON'T SAVE A PARTIAL FILE
       JMP       MENU

* WRITESK - WRITE A SINGLE 128 BYTE RECORD TO DISK *
WRITESK  LXI H,DSKBUFF  ;POINT TO DISK BUFFER
         CALL      DMASET    ;SET DMA ADDRESS
         CALL      WRITEREC  ;WRITE RECORD TO DISK
         CPI       0          ;GOOD WRITE?
         RZ          ;YES, DONE
         MVI       A,18       ;NO, OUT OF DISK SPACE
         CALL      ERROR      ;YES, DONE
         CALL      CLOSFILE   ;CLOSE THE FILE BUT
         CALL      DELETE     ;DON'T SAVE A PARTIAL FILE
         JMP MENU

* READREC - READ A SINGLE RECORD FROM DISK *
READREC  PUSH     B
         PUSH      H
         LXI       D,FCB
         MVI       C,READEF
         CALL      BIOS
         POP       H
         POP       L
POP B
RET

* WRITEREC - WRITE A SINGLE RECORD TO DISK *

WRITEREC PUSH B
PUSH D
PUSH H
LXI L, FCB
MVI C, WRITEF
CALL BDOS
POP H
POP D
POP B
RET

* LMASET - SET DMA ADDRESS
* CALL WITH ADDRESS IN HL
* RETURN WITH HL = HL + 128 *

LMASET PUSH PSW
PUSH B
PUSH L
PUSH H
XCHG ; DE = DMA ADDRESS
MVI C, SETIMA
CALL BDOS
POP D
LXI B, 128 ; READY DMA ADDRESS FOR NEXT
CALL B
POP D
POP B
POP PSW
RET

* OPENFILE - OPEN A FILE CURRENTLY ON DISK *

OPENFILE PUSH B
PUSH D
PUSH H
LXI L, FCB
MVI C, OPENF
CALL BDOS
POP H
POP D
POP B
RET

* CLOSFILE - CLOSE A FILE CURRENTLY ON DISK *

CLOSFILE PUSH B
PUSH D
PUSH H
LXI D,FCB
MVI C,CLOSEF
CALL BDOS
POP H
POP D
POP B
RET

* CREATE - CREATE A NEW FILE ON DISK *

CREATE PUSH B
PUSH L
PUSH H
LXI D,FCB
MVI C,MAKEF
CALL BDOS
POP H
POP L
POP B
RET

* DELETE - DELETE A FILE CURRENTLY ON DISK *

DELETE PUSH B
PUSH D
PUSH H
LXI D,FCB
MVI C,DELF
CALL BDOS
POP H
POP D
POP B
RET

* CURDSK - GET CURRENTLY LOGGED DISK *

CURDSK PUSH B
PUSH D
PUSH H
LXI D,FCB
MVI C,CURRENTL
CALL BDOS
POP H
POP D
POP B
RET

* ERROR - ERROR HANDLING ROUTINE
* CALL WITH ACC = ERROR NUMBER *

99
ERROR MOV C,A ;GET ERROR NUMBER
MVI B,0 ;COMPUTE ERROR VECTOR
LXI H,ERRJMP-3
DAD B
DAD B
LAL B ;ERROR VECTOR IS IN PC
PCHL
NOP
NOP

ERRJMP JMP ERROR1 ;MENU SELECTION ERROR
JMP ERROR2 ;TOO MANY/FEW DELIMITERS
JMP ERROR3 ;PERIOD+DATA ERROR
JMP ERROR4 ;INVALID HEX DIGIT ERROR
JMP ERROR5 ;DELIMITER AT START/END
JMP ERROR6 ;2 OR MORE DEL. SEQUENTIALLY
JMP ERROR7 ;NO AMBIGUOUS FILES
JMP ERROR8 ;COLONS NOT PROPERLY PLACED ; IN FILENAME
JMP ERROR9 ;TOO MANY/FEW CHAR. IN FN
JMP ERROR10 ;FEX FILETYPE ONLY
JMP ERROR11 ;NO SPACES IN FILENAME
JMP ERROR12 ;NO NON-PRINTABLE CHAR IN FN
JMP ERROR13 ;FILE NOT FOUND
JMP ERROR14 ;HEX CHECKSUM ERROR
JMP ERROR15 ;LISK READ ERROR
JMP ERROR16 ;OUT OF DIRECTORY SPACE
JMP ERROR17 ;START > FINISH ADDRESS
JMP ERROR18 ;OUT OF LIR/LISK SPACE ; PARTIAL FILE NOT SAVED

ERROR1 LXI C, MENERRMG
JMP ERROUT ;PRINT MENU ERROR MESSAGE

ERROR2 LXI C, MFILEERR
JMP ERROUT1 ;PRINT ERROR MESSAGE

ERROR3 LXI D, PERONLYM
JMP ERROUT1

ERROR4 LXI D, INVHEXER
JMP ERROUT1

ERROR5 LXI D, SEQDELER
JMP ERROUT1

ERROR6 LXI D, SEQDELER
JMP ERROUT1

ERROR7 LXI D, AMBIGERR

100
JMP ERROUT2
ERROR8 LXI E, COLONERR JMP ERROUT2
ERROR9 LXI D, FNCHARERR JMP ERROUT2
ERROR10 LXI E, HEXFTERR JMP ERROUT2
ERROR11 LXI E, SPFNEERR JMP ERROUT2
ERROR12 LXI D, NFRERR JMP ERROUT2
ERROR13 LXI D, FNFDERR JMP ERROUT1
ERROR14 LXI E, CKSUMERR JMP ERROUT3
ERROR15 LXI D, DSKRDERR JMP ERROUT3
ERROR16 LXI E, DIRSPERR JMP ERROUT
ERROR17 LXI E, SGFAERR JMP ERROUT1
ERROR18 LXI D, DDSPCERR JMP ERROUT3

ERROUT CALL PRINT ; PRINT ERROR
CALL DELAY ; LET USER READ ERROR
JMP MENU ; START OVER

ERROUT1 LXI SP, STACK CALL PRINT ; RE-INIT. STACK CALL PRINT ; PRINT ERROR
CALL DELAY ; RECALL MENU CHOICE
LDA MENUFLG ; RESTART CURRENT OPTION
JMP MENU1

ERROUT2 CALL PRINT ; PRINT ERROR
CALL DELAY ; BACK TO CALLER
RET

ERROUT3 CALL PRINT ; PRINT ERROR
CALL DELAY
CALL HOSTONE ; TELL MS DOME
JMP MENU

* DELAY - APPROX. 1-2 SECOND DELAY FOR USER TO SEE ERROR
* MESSAGE BEFORE MENU IS REPRINTED *

DELAY PUSH PSW
PUSH B
PUSH D
PUSH H
MVI B,15 ; OUTER LOOP INITIALIZATION
LXI E,-1 ; DECREMENT BY SUBTRACTION
DELAYIN LXI H,39E0H ; INNER LOOP INITIALIZATION
DELAYOUT DAD D ; HL = HL - 1
JC DELAYOUT
DCR B
JNZ DELAYIN
POP a
POP r
POP B
POP PSW
RET ; DELAY DONE, BACK TO CALLER

* CRLF - CARRIAGE RETURN & LINE FEED UTILITY *

CRLF MVI E, CR ; PRINT CARRIAGE RETURN
CALL CONSOUT
MVI E, LF ; THEN A LINE FEED
CALL CONSOUT
RET

* ENTER - GET A HEX INTEGER FROM THE CONSOLE BUFFER
* & RETURN WITH HL = 16 BIT BINARY DATA
* CALL WITH C = MAX NUMBER OF CHARACTERS TO INPUT
* LE = CONSOLE BUFFER POINTER FOR START OF
* CONVERSION PROCESS *

ENTER PUSH PSW ; SAVE A, BC, DE
PUSH B
PUSH D
LXI H, 0000H ; INIT. DATA AREA
ENTER1 LLAX D, A ; GET DATA FOR CONVERSION
CPI 0A' ; IS IT 0-9?
JC ENTER15 ; YES
ANI 0EFH ; NO - FORCE TO UPPER CASE
ENTER15 DAD H ; SHIFT PREVIOUS DATA LEFT
DAD H ; 4 BITS
DAD H
JC ENTER3 ; IF OVERFLOW, PRINT ERROR

102
CPI '0' ; IS IT 0-F?
JC ENTER3 ; NO - ILLEGAL CHARACTER
CPI 'F+1' ; IS IT > F?
JNC ENTER3 ; YES - ILLEGAL CHARACTER
CPI 'A' ; LEGAL - IS IT A-F?
JC ENTER2 ; NO - IT'S 0-9
ADI 9 ; ADD CONVERSION FACTOR
ENTER2 ANI 0FH ; ISOLATE 4 BITS
ORA L ; MERGE WITH PREVIOUS DATA
MOV L, A
DCR C ; COUNT CHARACTERS ENTERED
JZ ENTER4 ; EXIT IF C = 0
INX D ; BUMP BUFFER ADDRESS
JMP ENTER1 ; GET ANOTHER HEX INTEGER
ENTER3 MVI A, 4 ; PRINT ILLEGAL CHARACTER
JMP ERROR ; ERROR
ENTER4 POP C ; RESTORE REGISTERS
POP B
POP PSW
RET

* CONSIN - CONSOLE INPUT ROUTINE
* DOESN'T RETURN UNTIL INPUT IS RECEIVED *
CONSIN PUSH B ; SAVE REGISTERS
PUSH C
PUSH H
MVI C, CONIN ; GET CHARACTER
CALL BDOS
POP H
POP L
POP R
RET ; RETURN TO CALLER WITH CHARACTER IN A

* CONSOUT - CONSOLE OUTPUT ROUTINE
* ENTER WITH CHARACTER IN B *
CONSOUT PUSH PSW ; SAVE REGISTERS
PUSH B
PUSH C
PUSH H
MVI C, CONOUT ; OUTPUT CHARACTER
CALL BDOS
POP H
POP L
POP B
POP PSW
RET ; BACK TO CALLER
* CONSTAT - GET CONSOLE INPUT STATUS
* RETURNS WITH A = 00H IF NO CHARACTER WAITING
* = 0FFH IF CHARACTER IS WAITING *

CONSTAT
    PUSH B ; SAVE REGISTERS
    PUSH D
    PUSH H
    MVI C, CONST ; GET STATUS
    CALL BIOS
    POP H ; RESTORE REGISTERS
    POP C
    POP B
    RET

* BUFFRL - READ CONSOLE INPUT INTO BUFFER POINTED TO BY LE
* RETURN WITH DE = BUFFER START ADDRESS + 1
* B = COUNT OF CHARACTERS INPUT
* ALL OTHER REGISTERS (A, HL) UNCHANGED *

BUFFRL
    PUSH PSW ; SAVE A, HL
    PUSH H
    LXI E, PROMPT ; SEND PROMPT TO CONSOLE
    CALL PRINT
    LXI E, CONBUFF ; POINT TO CONSOLE BUFFER
    PUSH D ; SAVE IT
    MVI C, READCON ; READ CONSOLE INPUT
    CALL BIOS
    CALL CRLF
    POP D ; POINT TO CHAR. COUNT
    INX E
    LEAX D ; GET COUNT
    ORA A ; IS COUNT = 0?
    JNZ READONE ; NO, RETURN TO CALLER
    JMP BUFFRL ; YES, TRY AGAIN
    READONE
    MOV B, A ; RETURN WITH B = COUNT
    POP H ; RESTORE A, HL
    POP PSW
    RET

* BUFFRL1 - READ CONSOLE INPUT INTO BUFFER POINTED TO BY LE
* RETURN WITH DE = BUFFER START ADDRESS + 1
* B = COUNT OF CHARACTERS INPUT
* A = 00H IF COUNT = 0
* = 0FFH IF COUNT > 0
* HL UNCHANGED *

BUFFRL1
    PUSH H ; SAVE HL
    LXI D, CONBUFF ; POINT TO CONSOLE BUFFER
    PUSH D ; SAVE IT
    MVI C, READCON ; REAL CONSOLE INPUT
    CALL BIOS
CALL CRLF ;POINT TO CHAR. COUNT
POP L
INX L
LDAX D ;GET COUNT
ORA A ;IS COUNT = 0?
JZ READONE1 ;YES, RETURN TO CALLER
MOV B,A ;SAVE CHAR COUNT
MVI A,OFFH ;COUNT > 0
JMP READONE1+1
READONE1 MOV B,A ;RETURN WITH B = COUNT
POP H ;RESTORE A, HL
RET

* SCAN - DELIMITER SCAN OF CONSOLE INPUT BUFFER
* (SPACES AND COMMAS ARE LEGAL DELIMITERS)
* ALSO CHECKS FOR ESCAPE AND '?' KEYS
* CALL WITH DE = CONBUFF + 1
* A = NUMBER OF DELIMITERS TO LOOK FOR
* RETURN WITH CARRY SET IF MORE OR LESS DELIMITERS
* THAN SPECIFIED
* A = GARBAGE
* OTHER REGISTERS UNCHANGED *

SCAN PUSH B ;SAVE REGISTERS
PUSH L
PUSH H
MOV B,A ;GET DELIMITER COUNT
XCHG ;HL = CONBUFF + 1
MOV C,M ;GET CHARACTER COUNT
CALL SCNENDEL ;SCAN FOR DELIMITERS AT START AND END OF INPUT
CALL SCANDDEL ;SCAN FOR SEQUENTIAL DELS.
SCAN1 INX H ;GET CHARACTER
MOV A,M
CPI SPACE ;IS IT A SPACE?
JZ CNTDEL ;YES, DEC DELIMITER COUNT
CPI COMMA ;IS IT A COMMA?
JZ CNTDEL ;YES
CPI ESC ;IS IT AN ESCAPE CHARACTER?
JZ ESCESC ;YES, ESCAPE FROM OPTION
CPI '?' ;IS IT A QUEST FOR HELP?
JZ QUESTION ;YES, PRINT DATA FORMATS
SCAND1 CMP B ;NO MORE CHARACTERS TO CHECK -
JMP SCAN1 ;INCREMENT DELIMITER COUNT
CNOEL ECR B ;LOOK FOR ANOTHER DELIMITER
JMP SCAN2 ;SEE IF B = 0
SCANDONE XRA A
CMP B
SCAND1 POP H ;RESTORE REGISTERS
POP D
POP B
RET

SCANESC LEA MENUFLG ; IF HOST COMMANI THEN
CPI NHOSTCM ; NO ESCAPE TO MDS
JC MENU
LEA SYSSTAT ; SEE IF HOST IN CONTROL
ORA A
JZ SCNESCL ; HOST IN CONTROL
CALL CNTRLCK ; MDS IT IN CONTROL
RRC
JNC MENU ; NO ABORT

SCNESCL MVI A,'Q' ; ABORT
CALL MDCMD
XRA A ; CLEAR SYSSTAT FLAG, HOST
STA SYSSTAT ; NOW IN CONTROL
JMP MENU ; RETURN TO MENU

QUESTION LXI D,FROMTMSG ; PRINT DATA FORMATS AND
CALL PRINT ; RETURN TO CURRENT OPTION
QUEST1 CALL CONSTAT ; WAIT FOR RESPONSE TO
RRC
JNC QUEST1 ; CONTINUE
CALL CONSIN
LEA D,FMTMSG1 ; CONTINUE FORMAT MESSAGE
CALL PRINT

QUEST2 CALL CONSTAT
RRC
JNC QUEST2
CALL CONSIN
LEA MENUFLG
JMP MENU1 ; BACK TO OPTION

CNTRLCK LXI D,ABORTMSG ; MDS IS PRINT ABORT QUERY
CALL PRINT
CNTRL1 CALL CONSTAT ; WAIT FOR RESPONSE
RRC
JNC CNTRL1 ; GET RESPONSE
CALL CONSIN
ORI 20H ; FORCE IT TO LOWER CASE
CPI y' ; ABOET MDS CONTROL?
JZ CNTRL2 ; YES
XRA A ; NO, CLEAR A
RET

CNTRL2 MVI A,OFFH ; SET A
RET

* SCNENDEL - CHECK FOR DELIMITERS AT FIRST & LAST CHARACTER
* POSITIONS IN CONSOLE INPUT BUFFER
* CALL WITH BUFFER COUNT IN C *
* SCANDEL - SCAN CONSOLE BUFFER FOR 2 OR MORE SEQUENTIAL DELIMITERS *

SCANDEL PUSH B ;SAVE BUFFER COUNT
INX H ;GET FIRST CHARACTER
MOV A,M
CPI SPACE ;IS IT A SPACE?
JZ SCNSPC1 ;YES, ERROR
CPI COMMA ;IS IT A COMMA?
JNZ SCNSPC2 ;NO, CONTINUE TO END
SCNSPC1 MVI A,5 ;ERROR
JMP ERROR
SCNSPC2 DCR C ;AT BUFFER END YET?
JZ SCNSPC3 ;YES
INX H ;NO
JMP SCNSPC2 ;LOOP
SCNSPC3 MOV A,M ;GET LAST CHARACTER
CPI SPACE ;A SPACE?
JZ SCNSPC1 ;YES, ERROR
CPI COMMA ;A COMMA?
JZ SCNSPC1 ;YES, ERROR
POP B ;RESTORE BUFFER COUNT
LXI H,CONBUFF+1 ;AND POINTER TO IT
RET

SCANDEL PUSH B ;SAVE BUFFER COUNT
STA FRSTDEL ;INIT. FIRST DELIMITER FLAG
STEL1 INX H ;GET CHARACTER
MOV A,M
CPI SPACE ;SPACE?
JZ DELCK ;YES, FIRST DELIMITER?
CPI COMMA ;COMMA?
JZ DELCK ;YES, FIRST DELIMITER?
DCR C ;IF C = 0 THEN DONE
JZ SEELNE
XRA A ;RESET FLAG
STA FRSTEEL
JMP STEL1 ;LOOP
DELCK LDA FRSTDEL ;FIRST DELIMITER?
ORA A
JNZ DELCK1 ;NO, A=1 - ERROR
INR A ;YES, SET FRSTDEL FLAG
STA FRSTEEL
DCR C ;SEE IF DONE
JZ SEELNE
JMP STEL1 ;NO, LOOP
DELCK1 XRA A
STA FRSTDEL
MVI A,6
JMP ERROR ;PRINT ERROR
SDELDNE POP B ;RESTORE BUFFER COUNT
LXI H,CONBUFF+1 ; AND POINTER TO IT
RET

* CKPERIOD - CHECK FOR A PERIOD ANYWHERE IN INPUT
* CALL WITH DE = CONBUFF + I
* RETURN WITH A = 00 IF NO PERIOD FOUND
* = OFPH IF A PERIOD ONLY
* = OFPH IF A PERIOD + DATA
* OTHER REGISTERS UNCHANGED *

CKPERIOD PUSH B ;SAVE REGISTERS
PUSH L
PUSH H
XCHG
MOV C,M ;C = CHARACTER COUNT
MOV L,M ;L = CHAR. COUNT ALSO
CKPER1 INX H ;GET CHARACTER
MOV A,M
CPI PERIOD ;IS IT A PERIOD?
JZ PERFND ;YES
LCR C ;NO, ANY MORE CHARACTERS?
JZ CKDONE ;NO, CHECK DONE
JMP CKPER1 ;YES, TRY AGAIN
PERFND MOV A,L ;RECALL ORIG. CHAR. COUNT
CPI 1 ;ONLY A PERIOD?
JZ NOERR ;YES, NO ERRCR
MVI A,OFPH ; PERIOD + DATA IS ILLEGAL
JMP CKDONE+1
NOERR MVI A,OFPH ;PERIOD ONLY INDICATION
JMP CKDONE+1
CKDONE XRA A ;CLEAR ACC., NOT FOUND
POP H ;RESTORE REGISTERS
POP D
POP B
RET

* GET4BIN - GET 4 OR LESS HEX INTEGERS FROM THE CONSOLE
* BUFFER AND CONVERT THEM INTO 16 BIT BINARY DATA
* (GO INTO BUFFER, GC TO DELIMITER IF ONE EXISTS
* OR TO BUFFER END, WHICHEVER OCCURS FIRST;
* BACK UP NUMBER OF CHARACTERS SPECIFIED BY
* CALLER OR TO DELIMITER OR BUFFER+1, CONVERT
* TO BINARY AND RETURN)
* CALL WITH DE = START OF CONVERSION POINTER (AT A
* DELIMITER OR THE BUFFER COUNT)
* RETURN WITH B = NUMBER OF CHARACTERS LEFT IN BUFFER
* C = NUMBER OF CHARACTERS CONVERTED
* DE = END OF BUFFER OR DELIMITER
* HL = 16 BIT BINARY DATA *
GET4BIN MVI C,4 ; GET 4 CHARACTERS MAX
MOV A,C ; BE SURE BACKUP1 INST IS STA BACKUP1+1 ; MVI A,4
GET41 INC ;HL = START OF SEARCH
GET4LOOP INX H ; GET CHARACTER
MOV A,M
CPI SPACE ;IS IT A SPACE?
JZ BACKUP ; YES
CPI COMMA ;IS IT A COMMA?
JZ BACKUP ; YES
LCR B ;MORE CHARACTERS IN BUFFER?
JMP GET4LOOP ; NO
BACKUP0 INX H ; POINT TO BUFFER END + 1
BACKUP PUSH H ; SAVE DELIMITER ADDRESS
DCX H ; BACK UP 1
CALL BUFFST ; AT BEGINNING OF BUFFER?
JZ BACKUP01 ; NO
MOV A,M
CPI SPACE ; ARE WE AT A SPACE?
JZ BACKUP01 ; YES
CPI COMMA ; ARE WE AT A COMMA?
JZ BACKUP01 ; YES
LCR C ; DECREMENT CHARACTER COUNT
JNZ BACKUP+1 ; BACK UP 1 AGAIN
JMP BACKUP1 ; C = 0 FINALLY
BACKUP01 INX H ; POINT TO FIRST CHARACTER
BACKUP1 MVI A,4 ; FINALLY GOT THERE
SUB C ; COMPUTE NUMBER OF BACKUPS
MOV C,A
Xchg ; DZ = CONVERSION START ADDR
CALL ENTER ; TO CONVERSION
POP D ; IE = DELIMITER ADDRESS
DCR B ; DECREMENT CHAR. COUNT
RET
BUFFST PUSH H
PUSH D
LXI L,CONBUFF+1
MOV A,L ; AT BUFFER+1 YET?
CMP E ; IF Z = 1 THEN AT BUFFER+1
POP E
POP H
RET ; ELSE Z = 0

* GET2BIN - SAME AS GET4BIN BUT LIMITED TO TWO CHARACTERS MAX
* SAME ENTRY PARAMETERS
* RETURNS WITH L = 8 BIT BINARY DATA
* OTHER REGISTERS AS IN GET4BIN *
GET2BIN
MVI C,2
MOV A,C
STA BACKUP1+1
CALL GET41
MVI A,4
STA BACKUP1+1
RET

* MDSOUT - HOST OUTPUT TO MDS
* CALL WITH CHARACTER IN A *

MDOUT
PUSH B
PUSH C
PUSH H
MOV C,A
MDSOUT1
MVI A,10H
OUT MSTATPT
IN MSTATPT
ANI 0CH
CPI 0CH
JNZ MDSOUT1
MOV A,C
OUT MSTAP
CPI XON
JZ XONDN
XONCK CALL MESTAT
RRC
JNC XONCK
IN MSTAP
XCNNDN POP H
POP D
POP B
RET

* MISCMD - SEND COMMAND TO MDS
* CALL WITH A = COMMAND *

MDCMD
PUSH PSW
MVI A,059H
CALL MDSOUT
POP PSW
CALL MDSOUT
RET

* MDATAOUT - SEND USABLE DATA TO MDS
* CALL WITH A = DATA *

MDATAOUT
PUSH PSW
MVI A,0FFH
CALL MDSOUT
POP PSW

PUSH PSW ; SAVE IT
CALL MESOUT
POP PSW ; RESTORE DATA
RET

* HOSTRELY - HOST READY TO RECEIVE RETURN DATA FOR CURRENT
   OPTION *

HOSTRELY MVI A,00H ; NEXT CHAR. IS RDY FLAG
CALL MESOUT
MVI A,00H ; SEND READY FLAG
CALL MDSOUT
RET

* HOSTDONE - HOST DONE WITH ITS PART IN CURRENT OPTION,
   IS RETURNING TO MONITOR *

HOSTDONE MVI A,'Q' ; NEXT CHAR. IS DONE CMD
CALL MDSCMD
RET

* MISIN - HOST INPUT FROM MDS
   * RETURNS WITH CHARACTER IN A, OTHER REGISTERS RESTORED *

MISIN PUSH B ; SAVE REGISTERS
PUSH D
PUSH H
CALL MESINRELY ; ANY INPUT WAITING FROM MDS?
IN MDATAPT ; YES, GET DATA TYPE
CPI 0FFH ; IS IT DATA?
JZ MISIN2 ; YES, GET IT
CPI 055H ; QUIT CMD?
JZ MESQUIT ; YES
JMP MESINLINE ; NO, MDS MUST HAVE
   SIGNALLED IT'S READY
   ; FOR INPUT
MISQUIT MVI A,'XON' ; CONFIRM RECEIPT
CALL MDSOUT
CALL MESINRELY
IN MDATAPT
XRA A ; RESET FLAGS
STA SYSSTAT
STA MSLRYF
MVI A,'XON' ; CONFIRM RECEIPT OF 'Q'
CALL MESOUT
JMP MENU ; NOW BACK TO MENU
MISIN2 MVI A,'XON' ; SEND CONFIRMATION
CALL MDSOUT
CALL MESINRELY ; WAIT FOR DATA
IN MDATAPT ; THEN GET IT
PUSH PSW ; SAVE IT

111
MVI A,XON ; CONFIRM AGAIN
CALL MDSOUT
POP PSW ; RESTORE DATA & REGISTERS
POP B
POP D
POP H
RET

* MDSINRDY - CHECK FOR INPUT FROM MDS, LOOP TILL THERE IS *

MDSINRDY CALL ESCK ; CHECK FOR ESCAPE
CALL MDSTAT ; GET STATUS
RRC
JNC MDSINRDY ; NO CHARACTER WAITING, LOOP
RET ; CHARACTER WAITING

* MESINDNE - SET MDS READY FOR INPUT FLAG *

MESINDNE MVI A,XON ; CONFIRM IT
CALL MESOUT
CALL MESINRDY
IN MDATAPT
MVI A,OFPH ; SET MDS READY FLAG
STA MESRDYF
MVI A,XON ; CONFIRM RECEIPT OF DATA
CALL MESOUT
POP H ; RESTORE REGISTERS
POP D
POP B
RET ; BACK TO MESIN CALLER

* ESCK - CHECK FOR ESCAPE COMMAND FROM KEYBOARD
* IGNORE ALL OTHER INPUT *

ESCK CALL CONSTAT ; CHECK FOR INPUT
RRC
RNC ; NONE
CALL CONSLN ; IS IT ESCAPE?
CPI ESC ; IS IT ESCAPE?
JZ ESCK1 ; NO
MVI E,BKSPCE ; DON'T PRINT CHARACTER
CALL CONSLOUT
RET

ESCK01 LEA SYSSTAT ; GET SYSTEM STATUS
ORA A
JZ ESCK1 ; HOST IN CONTROL
CALL CNTRLCK ; SEE WHO IS IN CONTROL
RRC
JNC MENU ; NO ABORT
ESCK1 MVI A,'Q' ; YES, SEND ESCAPE CMD
CALL MDSCMD

112
XRA A
;HOST NOW IN CONTROL
STA SYSSTAT
JMP MENU
;NOW BACK TO MENU

* MESTAT - GET STATUS OF MES SIO
* RETURNS WITH A = 00 AND Z = 1 IF NO CHARACTER WAITING
* = 0FFH AND Z = 0 IF CHARACTER WAITING *

MSTAT XRA
OUT MSTATPT
IN MSTATPT
ANI 1
RZ
MVI A,0FFH
RET

* CNVT16 - CONVERT 16 BITS BINARY DATA TO HEX ASCII
* CALL WITH HL = ADDRESS FOR 4 CHARACTER ASCII OUTPUT
* STRING
* RETURNS REGISTER PAIRS UNCHANGED
* A = GARBAGE *

CNVT16 PUSH H
PUSH G
PUSH B
INX H
INX H
MVI D,4
;CHARACTER COUNTER
CNVT161 MOV A,C
ANI 0FH
CPI 0AH
JC CNVT1615
;IS IT A-F?
AD1 7
;YES
CNVT1615 AD1 '0'
;FORM ASCII
MOV M,A
;STORE THIS CHARACTER
DCX H
;BACK UP THROUGH OUTPUT AREA
MVI E,4
;DOUBLE RIGHT
ORA A
;SHIFT RIGHT 4 BITS
CNVT162 MOV A,B
RAB
MOV B,A
MOV A,C
RAB
MOV C,A
LCR E
;DECREMENT SHIFT COUNTER
JNZ CNVT162
;STILL SHIFTING
LCR I
;DECREMENT CHARACTER COUNTER
JNZ CNVT161
;STILL CONVERTING

113
POP B ;RESTORE REGISTERS
POP L
POP H
RET

* CNVT8 - CONVERT 8 BITS BINARY DATA TO HEX ASCII
* CALL WITH HL = ADDRESS FOR 2 CHARACTER ASCII OUTPUT
* STRING
* C = 8 BIT BINARY DATA
* RETURNS REGISTER PAIRS UNCHANGED
* A = GARBAGE *

CNVT8 PUSH H ;SAVE REGISTERS
PUSH D
PUSH B
INX H
MVI D,2
JMP CNVT161 ;DO CONVERSION

* STAR - PRINT A STAR *

STAR PUSH D
LXI E,STARMG ;PRINT IT
CALL PRINT
POP D
RET ;BACK TO CALLER

*** MISCELLANEOUS MESSAGE AND DATA STORAGE AREAS ***

SIGNON DB CR,LF,"ALTOS MES CONTROL PROGRAM"
DB " - VERSION 1.5",CR,LF,LF "$"

INSTRUC DB CR,LF,"BASIC AMDS INSTRUCTIONS:",CR,LF,LF
DB "A. THE PROMPT FOR INPUT OF DATA IS"
DB ">",CR,LF
DB "B. ALL INPUTS MAY BE IN UPPER OR LOWER CASE.",CR,LF
DB "C. ADDRESS AND DATA INPUTS ARE EXPECTED TO BE IN HEX NOTATION.",CR,LF
DB "D. TERMINATE INPUTS WITH A CARRIAGE RETURN OR LINE FEED.",CR,LF
DB "E. NORMAL LINE EDITING ON INPUT IS AS IN CP/M AND MP/M.",CR,LF
DB "F. FOR ADDRESS INPUTS, THE PROGRAM WILL ALWAYS TAKE THE LAST FOUR OR LESS HEX CHARACTERS ENTERED; FOR DATA INPUTS, THE LAST TWO OR LESS",CR,LF
DB "G. SOURCES OF COMMON ERROR ARE INVALID HEX DIGITS, TOO MANY OR TOO FEW DELIMITERS, AND ILLEGAL SYNTAX.",CR,LF
H. IN GENERAL, THE SAME DATA I/O FORMAT
AS USED IN DIGITAL RESEARCH'S CR,LF
TILL IS USED HERE. FOR EXCEPTIONS,
CONSULT THE USER'S MANUAL', CR,LF
I. A QUESTION MARK ENTERED AFTER THE
PROMPT WILL CAUSE THE INPUT FORMATS TO
BE DISPLAYED", CR,LF
J. IF THE ESCAPE KEY IS ENTERED DURING
INPUT THEN THE USER IS RETURNED', CR,LF
TO THE MENU', CR,LF
K. FOR FURTHER DETAILS, CONSULT THE
USER'S MANUAL', CR,LF, LF
PRESS ANY KEY TO CONTINUE >$

MENU MSG CR, LF
" " MENU', CR, LF
" " HOST COMMANDS
" " MDS COMMANDS', CR, LF, LF
" A. SUPPRESS PRINTING MENU
" G. DOWNLOAD HEX FILE - Disk TO MDS
" MEMORY', CR, LF
" B. DO NOT SUPPRESS PRINTING MENU
" H. UPLOAD MDS MEMORY TO HEX DISK FILE'
" CR, LF
" C. BASIC INSTRUCTIONS
" I. EXAMINE/SET MDS MEMORY LOCATION(S)
" CR, LF
" D. HEXADECIMAL ADD & SUBTRACT
" J. CONTINUOUS SET OF MDS MEMORY', CR, LF
" E. RETURN SYSTEM CONTROL TO HOST
" K. FILL MDS MEMORY WITH SPECIFIED BYTE'
" CR, LF
" F. RETURN TO CP/M
" L. LOCATE BYTE SEQUENCE IN MDS MEMORY'
" CR, LF
" M. DUMP MDS MEMORY LOCATION(S) TO CONSOLE'
" CR, LF
" N. EXECUTE MDS MEMORY FROM SPECIFIED', CR, LF
" LOCATION', CR, LF, '$'
" SYSTEM STATUS: $$$$ IN CONTROL;
" $$$ MENU SUPPRESSION', CR, LF, '$'
" MDS $'
" HOST$'
" NO$'
" $'
" INVALID MENU SELECTION', CR, LF, '$'
" TOO MANY OR TOO FEW DELIMITERS IN
"INPUT", CR, LF, 
PERONLYM DB CR, LF, "PERIOD ONLY PLEASE!", CR, LF, 
INVHEXER DB CR, LF, "INVALID HEX DIGIT!", CR, LF, 
SEDSLEER DB CR, LF, "CAN'T HAVE A DELIMITER AT START OR 
END OF INPUT!", CR, LF, 
SEQUELER DB CR, LF, "TWO OR MORE DELIMITERS SEQUENTIALLY!", CR, LF, 
AMBIGERR DB CR, LF, "AMBIGUOUS FILENAMES NOT ALLOWED!", CR, LF, 
COLONERR DB CR, LF, "COLON (: ) NOT PROPERLY PLACED IN 
FILENAME!", CR, LF, 
FNCHARER DB CR, LF, "FILENAME TOO LONG OR TOO SHORT!", CR, LF, 
HEXFTERR DB CR, LF, "HEX FILETYPES ONLY!", CR, LF, 
SPFNERR DB CR, LF, "NO SPACES ALLOWED IN FILENAME!", CR, LF, 
NFRTERR DB CR, LF, "NON-PRINTABLE CHARACTERS NOT 
ALLOWED IN FILENAME!", CR, LF, 
FNPNCERR DB CR, LF, "FILE NOT FOUND!", CR, LF, 
CKSUMERR DB CR, LF, "HEX CHECKSUM ERROR!", CR, LF, 
LSKRDERR DB CR, LF, "DISK READ ERROR!", CR, LF, 
DIRSPERR DB CR, LF, "OUT OF DIRECTORY SPACE!", CR, LF, 
SGFAERR DB CR, LF, "START ADDRESS CANNOT BE GREATER 
THAN FINISH ADDRESS!", CR, LF, 
LDSPCERR DB CR, LF, "OUT OF DIRECTORY OR DISK STORAGE 
SPACE!", CR, LF, 
DRIVERR DB CR, LF, "WARNING - ONLY CURRENTLY SELECTED 
DISK WILL BE USED, INPUT IGNORED!", CR, LF, 
CNTRLMSG DB CR, LF, "MDS IS IN CONTROL, CAN'T CONTINUE 
UNTIL OPTION 'E' IS SELECTED!", CR, LF, 
ABORTMSG DB CR, LF, "ABORT MDS CONTROL (Y/N)?!", CR, LF, 
ABORTEDM DB CR, LF, "MDS CONTROL ABORTED, HOST IN 
CONTROL!", CR, LF, 
EXMSG DB CR, LF, "WILL CONSOLE BE RECEIVING DATA 
FOR DISPLAY FROM THE MDS (Y/N)?!", CR, LF, 
EXMSG2 DB CR, LF, "MDS IS IN CONTROL, HOST MAY REGAIN 
CONTROL ONLY BY TYPING THE ESCAPE KEY!", CR, LF, 
FORMTMSG DB CR, LF, "INPUT PARAMETER FORMATS ARE AS 
FOLLOWS: ": CR, LF, 
"X IS OPTION SELECTION (A-N)!", CR, LF, 
"HEXARITH ": XXXX YYYY, CR, LF, 
"XXXX & YYYY ARE HEX INTEGERS!", CR, LF, 
"DOWNLOAD ": FILENAME(.HEX), CR, LF, 
"(.HEX) IS OPTIONAL!", CR, LF, 
"UPLOAD ": FILENAME(.HEX), CR, LF, 
"XXXX YYYY
' XXXX & YYYY ARE MDS HEX START AND ',CR,LF
' END ADDRESSES FOR UPLOAD ',CR,LF
' EXAMINE MDS >XXXX
' XXXX IS FIRST MDS HEX ADDRESS TO
' CR,LF
' EXAMINE AND SET ',CR,LF
' >XXXX YYYY ZZ
' XXXX IS HEX ADDRESS, YYYY IS HEX DATA
' CR,LF
' AT THAT ADDRESS, ZZ IS CARRIAGE RETURN
' CR,LF, or ZZ IS NEW HEX DATA
' CR,LF,
' or ZZ IS '...','CR,LF
' CONTINUOUS >XXXX
' XXXX IS MDS HEX START ADDRESS FOR
' CR,LF
' FIRST CHANGE ',CR,LF
' >AAAA BB CC ....
' ARE HEX DATA FOR ENTRY INTO MDS MEMORY
' CR,LF
' (255 ENTRIES MAX, INCLUDING DELIMITERS)
' CR,LF
' IF ONLY A '... ' IS TYPED AFTER THE
' CR,LF,
' PROMPT, THE OPTION IS END ',CR,LF
' FILL >XXXX YYYY ZZ
' XXXX & YYYY ARE MDS HEX START AND
' CR,LF
' END ADDRESSES TO FILL BETWEEN ',CR,LF
' ZZ IS HEX DATA TO USE FOR FILL ',CR,LF
' CR,LF, 'PRESS ANY KEY TO CONTINUE >$'
' CR,LF,LF
' LOCATE SEQ. >XXXX YYYY)
' XXXX & YYYY ARE MDS HEX START AND ',CR,LF
' CR,LF
' OPTIONAL END ADDRESSES TO SEARCH BETWEEN
' CR,LF
' >AAAA BB ..., PP
' ARE UP TO 16 BYTE HEX SEQUENCE ',CR,LF
' TO SEARCH FOR IN MDS MEMORY ',CR,LF
' CR,LF
' OPTIONAL END ADDRESSES TO LUMP BETWEEN'
' CR,LF
' CR,LF
' EXECUTE >XXXX
' XXXX IS MDS HEX ADDRESS WHERE EXECUTION'
DB CR,LF
LB ' IS TO BEGIN',CR,LF,LF
HEXMSG DB CR,LF,'HEX ALL/SUB',CR,LF,'$
HEXMSG1 DB 'SUM = $$',CR,LF,LF
HEXMSG2 DB 'DIFF = $$',CR,LF,LF
EXAMSG DB CR,LF,'EXAMINE/SET MDS MEMORY',CR,LF,'$
EXAMSG1 DB '$$$',CR,LF,LF
EXAMSG2 DB '$',CR,LF,LF
FILLMSG DB CR,LF,'FILL MDS MEMORY LOCATION(S)',CR,LF,LF
CSETMSG DB CR,LF,'CONTINUOUS SET MDS MEMORY W/O '
EXECMSG DB CR,LF,'EXECUTE MDS MEMORY FROM SPECIFIED '
LOCMSG DB CR,LF,'LOCATE BYTE SEQUENCE IN MDS MEMORY'
NOTFOUND DB CR,LF,'BYTE SEQUENCE NOT FOUND !',CR,LF,LF
FOUNDMSG DB CR,LF,'FOUND STARTING AT MDS ADDRESS '
FOUNDMS1 DB '$$$',CR,LF,LF
LUMP MSG DB CR,LF,'DUMP MDS MEMORY',CR,LF,LF
LUMPMSG1 DB '$$$',CR,LF,LF
LUMPMSG2 DB '$',CR,LF,LF
LUMPMSG3 DB '$$$ $$ $$ $$ $$ $$ $$ $$ $'
MENUPRO1 DB CR,LF,'OPTION A = MENU SUPPRESSION, B = '
DB 'NO MENU SUPPRESSION'
MENUPROM DB CR,LF,'INPUT MENU OPTION $'
DB '>',CR,LF,LF
FILENAME DB 'FILENAME $'
DWNLDMSG DB CR,LF,'DOWNLOAD HEX FILE FROM DISK TO MDS'
DB 'MEMORY',CR,LF,LF
DWNLDONE DB CR,LF,'DOWNLOAD COMPLETED',CR,LF,LF
DWNLDONE1 DB 'MDS START ADDRESS = $$H',LAST ADDRESS '
DB '=' $$H',CR,LF,LF
UPLDMSG DB CR,LF,'UPLOAD (SAVE) MDS MEMORY TO DISK '
DB 'HEX FILE',CR,LF,LF
UPLDONE DB CR,LF,'UPLOAD TO DISK SUCCESSFULLY '
DB 'COMPLETED',CR,LF,LF
DATAMSG DB '$$',CR,LF,LF
STARMG DB '*$',CR,LF,LF
DB 'HEX',0,0,0,0

SYSSTAT IS 1 ; SYSTEM STATUS FLAG
; HOST IN CONTROL = 0
; MDS IN CONTROL = 1

MENUSUPPF IS 1 ; MENU SUPPRESSION FLAG
; 0 = NO SUPPRESSION
; 1 = SUPPRESSION
; STORAGE FOR MENU CHOICE
FIRSTDEL DB 0
; FIRST DELIMITER FLAG
FIRST DW 0
; FIRST NUMBER TO ADD/SUB
SECOND EW 0
; SECOND NUMBER TO ADD/SUB
SUM DW 0
; SUM OF HEX NUMBERS
START DW 0
; STARTING ADDRESS FOR COMMANE USE
FINISH DW 0
; FINISH ADDRESS FOR COMMANE USE
MESDATA ES 1
; TEMP. STORAGE FOR DATA FROM MIS
CONSDATA DS 1
; TEMP. STORAGE FOR DATA FROM CONSOLE TO MIS
MESRDYF ES 1
; MIS READY FLAG
0FFH = DONE, 0 = NOT DONE
FIRSTIME ES 1
; FIRST TIME THROUGH REAL OPERATIONS
BUFFCNT ES 1
; BUFFER COUNT SPACE
CURRENT DS 1
; CURRENT DISK DRIVE
CONTFLG DS 1
; CONTINUATION FLAG FOR DISK READ OPERATIONS
00 = NC CONTINUE
0FFH = CONTINUE
FCB DS 36
; SPACE FOR FILE CONTROL BLOCK
CONBUFF DB 48
; DEFAULT TO 48 CHARACTERS MAX FOR CONSOLE BUFFER
DS 256
; PROVIDE FOR 255 CHARACTERS
LSKBUFF EQU $
; START OF DISK BUFFER
ENL STARTER
APPENDIX D

MDS MONITOR SOFTWARE LISTING

*******************************************************************************
*                  MDS MONITOR SOFTWARE LISTING                           *
*                                                                       *
* AMIS1 - ALTOS MICROCOMPUTER DEVELOPMENT SYSTEM                        *
* (MDS COLE)                                                            *
*                                                                       *
* VERSION 1.3, 28 MAY 1981                                              *
* LT. STEPHEN M. HUGHES - AUTHOR                                       *
*                                                                       *
* THIS IS THE MDS MONITOR COLE FOR THE AMDS. THE AMIS USER'S MANUAL     *
* SHOULD BE CONSULTED FOR SPECIFICS NOT GIVEN IN THE DOCUMENTATION      *
* WHICH FOLLOWS.                                                        *
*                                                                       *
*******************************************************************************

RAM EQU 2000H ;START OF ONBOARD RAM
CHASTAT EQU 0E4H ;CHANNEL A STATUS AND COMMAND/CONTROL PORT
CHADATA EQU 0E3H ;CHANNEL A DATA PORT
CHESTAT EQU 0E2H ;CHANNEL B STATUS AND COMMAND/CONTROL PORT
CHEDATA EQU 0E1H ;CHANNEL 3 DATA PORT ; (NOT USED IN THIS COLE)
BAUDREG EQU 0E0H ;PORT FOR SETTING BAUD RATE OF SERIAL PORTS
XON EQU 011H ;CONTROL Q

ORG 0000H ;START OF PROM
JMP PORTSET ;SET UP SERIAL PORT ON RESET
NOP
NOP
USERIO JMP USRIO ;USER CALL FOR CONSOLE I/O

ORG 0038H ;RST 7 LOCATION
JMP EXECINE ;USER RST 7 COMES HERE FOR RETURN OF CONTROL TO HOST;
; AND ONBOARD MONITOR

ORG 0040H ;RST 7 + 8
MCNITOR LXI SP,STACK ;SET STACK EVERY TIME

120
XRA A
STA OPTION ;RESET OPTION FLAG
CALL HSTIN ;GET COMMAND FROM HOST
MONITOR AND 7F
CPI 'W' ;COMMAND WILL BE ASCII
JZ DWNLD ;DOWNLOAD COMMAND?
CPI 'U' ;UPLOAD COMMAND?
JZ UPLD ;EXAMINE/SET MEMORY CRT?
CPI 'X' ;CONTINUOUS MEMORY SET CMI?
JZ EXAM ;FILL COMMAND?
CPI 'C' ;LOCATE SEQ. COMMAND?
JZ DUMP ;FILL MEMORY COMMAND?
CPI 'L' ;EXECUTE MEMORY CMI?
JZ LOCATE ;EXECUTE MEMORY COMMAND?
JZ DUMP ;EXECUTE MEMORY COMMAND?
JZ EXEC ;EXECUTE MEMORY CMI?
JMP MONITOR ;ANYTHING ELSE IS IGNORED

* DWNLD - DOWNLOAD HEX DISK FILE TO MTS MEMORY ROUTINE *
* ROUTINE LOOPS UNTIL A HOSTLINE COMMAND IS            *
* DETECTED BY THE INPUT ROUTINE *                        *

DWNLD CALL HOSTIN ;GET NUMBER OF BYTES TO ;EXPECT
        MOV C,A ;C = BYTE COUNTER
DWNLD1 CALL GETADDR ;GET STARTING ADDRESS
        MOV M,A ;STORE IT
        INX D ;MORE BYTES TO GET
        DCR C
        JNZ DWNLD1 ;GET NEW ADDRESS FIRST
        JMP DWNLD ;GET NEW ADDRESS FIRST

* UPLD - UPLOAD MTS MEMORY TO DISK HEX FILE *           *

UPLD CALL GETADDR ;GET STARTING ADDRESS
        SHLL START ;GET FINISH ADDRESS
        CALL GETADDR
        SHLL FINISH
        LHLL START
        XCHG ;DE = START ADDRESS
        LEAX L ;GET DATA
        INX D ;SEND IT
        CALL BUFFCMP ;DONE YET?
        JNC UPLD1 ;NO
CALL  MESRY  ; YES
JMP  MONITOR

* EXAM - EXAMINE/SET MEMORY
* LOOPS TILL INPUT DETECTS HOSTDONE COMMAND *

EXAM  CALL  GETADDR  ; GET STARTING ADDRESS
EXAM1  MOV  A,M  ; SEND DATA AT HL ADDRESS
          ; TO HOST
CALL  HEATAUT  ; GET NEW DATA
CALL  HOSTIN  ; DEPOSIT IT
INX  H
JMP  EXAM1  ; LOOP TILL HOSTDONE

* CSET - CONTINUOUS SET OF ME$ MEMORY
* LOOPS TILL HOSTDONE DETECTED *

CSET  CALL  GETADDR  ; GET STARTING ADDRESS
CSET1  CALL  HOSTIN  ; GET DATA
        MOV  M,A  ; DEPOSIT IT
        JMP  CSET1  ; LOOP

* FILL - FILL DESIGNATED MEMORY LOCATIONS WITH SPECIFIED DATA *

FILL  CALL  GETADDR  ; GET FIRST ADDRESS
SHLD  START
CALL  GETADDR  ; GET LAST ADDRESS
SHLD  FINISH
CALL  HOSTIN  ; GET DATA TO FILL WITH
MOV  C,A  ; SAVE IT
LELD  START
XCHG
FILL1  MOV  A,C  ; GET FILL DATA
STAX  D  ; DEPOSIT IT
INX  E
CALL  BUFFCMP  ; DONE YET?
RRC
JNC  FILL1  ; NO, KEEP FILLING
CALL  ME$DONE  ; YES
JMP  MONITOR

* LOCATE - LOCATE BYTE SEQUENCE IN ME$ MEMORY
* SENDS 'F' TO HOST IF FOUND
* SENDS 'N' TO HOST IF NOT FOUND *

LOCATE  CALL  GETADDR  ; GET START ADDRESS
SHLD  START
CALL  GETADDR  ; GET FINISH ADDRESS
SHLD  FINISH
LXI H, DATABUFF ; STORE SEQUENCE HERE
MVI C, 0 ; DATA COUNTER
CALL HOSTIN ; GET SEQUENCE
PUSH PSW
LDA HSTRDYFL ; IF SET THEN NO MORE DATA
RRC
JC SEARCH ; START SEARCH
POP PSW ; MORE DATA
MOV M, A ; STORE IT
INX H
INR C ; BUMP COUNTER
JMP LOCIN
SEARCH MOV A, C ; GET SEQUENCE COUNT
STA LOCOUNT ; SAVE IT
LHLE
XCHG ; LE = START ADDRESS
LXI E, DATABUFF ; RL = START OF SEQUENCE
SRCH1 LEAX E ; GET MORE DATA
CMP M ; IS THERE A MATCH?
JZ MATCH ; YES
INX E ; NO, SEE IF DONE
CALL BUFFCMP
RRC
JC NOTFNL ; YES, SEQ. NOT FOUND
JMP SRCH1 ; NO, TRY AGAIN
MATCH SHLL MATCHALR ; SAVE IT
XCHG ; RESTORE LE & RL
MATCH1 DCR C ; ALL MATCHES YET?
JZ FOUNI ; YES, FOUND SEQUENCE
INX E ; NO, START ALL OVER
CALL BUFFCMP ; ONE YET?
RRC
JC NOTFNL ; YES, SEQ. NOT FOUND
INX E ; NO, LOOK FOR NEXT MATCH
LLAX E
CMP M ; ANOTHER MATCH?
JZ MATCH1 ; YES
LHLE DATABUFF ; NO, START ALL OVER
INX E ; RE-INIT. SEQ. COUNT
LDA LOCOUNT
MOV C, A
JMP SRCH1 ; KEEP TRYING
FOUND MVI A, 'F' ; SEND FOUND TO HOST
CALL HDATAOUT
LHLE MATCHALR ; GET FIRST ADDR. OF MATCH
MOV A, H ; SEND IT TO HOST, MSB FIRST
CALL HDATAOUT
MOV A, L ; THEN LSB
CALL HDATAOUT
JMP MONITOR ; ALL DONE
NOTFND MVI A, 'N' ; SEND NOT FOUND TO HOST
CALL HDATAOUT
JMP MONITOR

* DUMP - DUMP MLS MEMORY TO HOST CONSOLE *

LUMP CALL GETADDR ; GET START ADDRESS
SHLD START
CALL GETADDR ; GET FINISH ADDRESS
SHLD FINISH
LHLD START
XCHG D ; DE = START ADDRESS
LUMP1 LLAX D ; GET MLS MEMORY DATA
INX D
CALL BUFFCMP ; DONE YET?
RRC JNC DUMP1 ; NO
CALL MESSRY ; YES
JMP MONITOR

* EXEC - EXECUTE MLS MEMORY *
* PROGRAM TO BE EXECUTED MAY RETURN MONITOR VIA *
* A "REST 7" INSTRUCTION OR A JUMP TO LOCATION *
* 0000H *
* HOST CONSOLE I/O IS AVAILABLE AS EXPLAINED IN *
* THE USRIO ROUTINE *

EXEC STA OPTION ; SAVE OPTION
CALL GETADDR ; GET EXECUTION ADDRESS
PCHL ; GO TO IT

*** UTILITY SUBROUTINES ***

* BUFFCMP - COMPARE LE TO FINISH ADDRESS + 1 *
* IF EQUAL, RETURN A = OFFH *
* IF UNEQUAL, RETURN A = 00 *

BUFFCMP PUSH H ; DE = CURRENT ADDR TO COMPARE
PUSH D
LHLD FINISH ; HL = FINISH ADDRESS + 1
INX H
MOV A, H ; R = D?
CMP L
JNZ NOCMP ; NO
MOV A, L ; YES, L = E?
CMP E
JNZ NOCMP ; NO
MVI A, OFFH ; YES, ADDRESSES ARE EQUAL
POP D

124
POP H
RE:
NOCMP XRA A
POP POP H
RET

* GETADDR - GET ADDRESS FROM HOST *

GETADDR CALL HOSTIN
MOV H,A
CALL HOSTIN
MOV L,A
RET

* PORTSET - SET UP SERIAL I/O PORTS ON EVERY RESET OR CALL TO 0000H *

PORTSET MVI A,77H
OUT BAUDREG
MVI A,01011110B
OUT CHASTAT
OUT CHESTAT
MVI A,00110111B
OUT CHASTAT
OUT CHESTAT
JMP MONITOR

* USRIO - USER TO/FROM HOST CONSOLE I/O ROUTINE
* USER EXECUTED PROGRAMS IN MES MEMORY MAY
* COMMUNICATE WITH THE HOST CONSOLE VIA A CALL
* TO LOCATION 0005H
* - FOR INPUT FROM THE HOST CONSOLE, CALL WITH
* REG. C = 1 - CHARACTER WILL BE RETURNED IN A
* - FOR OUTPUT TO HOST CONSOLE, CALL WITH THE
* CHARACTER IN A ANE REG. C = 2
* - TO CHECK FOR HOST INPUT, CALL WITH
* REG. C = 3 - RETURNS A = 00 IF NO INPUT HAS BEEN
* RECEIVED FROM THE HOST; A = 0FFF IF INPUT IS
* WAITING
* - IF C <> 1, 2 OR 3 THEN ROUTINE RETURNS WITH C = 0FFF

USRIO PUSH PSW
MOV A,C
CPI 1
JZ USRIN
CPI 2
JZ USROUT
CPI 3
CZ HOSTAT
JMP \_ 

125
MVI C, OFFH ; ILLEGAL CODE
RET

USRIN CALL MISRDY ; TELL HOST TO SEND INPUT
POP PSW
CALL HOSTIN ; GET INPUT
RET

USROUT POP PSW
CALL HTATAOUT ; SEND CHARACTER TO HOST
RET

* EXECDE - THIS RETURNS USER PROGRAM TO MONITOR AND
* RETURNS CONTROL TO HOST IF A RST 7 IS EXECUTED *

EXECDE LDA OPTION ; SEE IF THE EXECUTE OPTION
CPI 'E' ; WAS IN EFFECT WHEN CONTROL
JNZ MONITOR ; WAS TRANSFERRED HERE
CALL MISRDY ; NO, HOST IN CONTROL
JMP MONITOR ; YES, GIVE HOST CONTROL

* HOSTIN - GET INPUT FROM HOST & INTERPRET TYPE OF INPUT *

HOSTIN CALL GETCHAR ; GET INPUT
HOSTIN1 CPI 55H ; IS IT A COMMAND?
JZ HOSTCMC
CPI OFFH ; IS IT DATA?
JZ HOSTDTA
JMP HOSTRDY ; MUST BE HOST READY FLAG
HOSTCMC CALL GETCHAR ; GET ACTUAL COMMAND
JMP MONITOR1 ; GO TO MONITOR FOR DECODE
HOSTDTA CALL GETCHAR ; GET DATA
RET ; RETURN TO CALLER WITH IT
HOSTRDY CALL GETCHAR ; GET READY FLAG
MVI A, OFFH ; SET FLAG IN MLS
STA HSTRDYFL
RET ; RETURN TO CALLER

GETCHAR CALL HOSTAT ; LOOP TILL CHAR. IS WAITING
RRC
JNC GETCHAR
GETCHAR1 IN CHDATA ; GET DATA
PUSH PSW
MVI A, XON
CALL HOSTOUT ; CONFIRM IT
POP PSW
RET

* HOSTOUT - SEND DATA TO HOST *

HOSTOUT PUSH PSW
CALL HOSTAT ; ANYTHING FROM HOST? (HOST
RRC  HOSTOUT1  ; HAS PRIORITY
JNC  GETCHAR1  ; NO
CALL  HOSTIN1  ; YES, GET IT
CALL  ; IF COMMAND, BACK TO MONITOR

HOSTOUT1 IN  CHASTAT  ; GET PORT STATUS
ANI  1
JZ  HOSTOUT1  ; LOOP TILL READY TO SEND
POP  PSW  ; SEND CHARACTER
OUT  CHADATA  ; DON'T WAIT FOR XON
CPI  XON  ; CONFIRMATION
RZ  XONCK  CALL  HOSTAT  ; WAIT FOR CONFIRMATION
RRC
JNC  XONCK
IN  CHADATA  ; GET IT
RET

* HOSTSTAT - HOST INPUT STATUS *

HOSTAT IN  CHASTAT
ANI  2
RZ
MVI  A,0FFH
RET

NO CHAR. WAITING, RET A=0
CHAR. WAITING, RET A=0FFH

* HLDATAOUT - SEND DATA TO HOST IN PROPER FORMAT *

HLDATAOUT PUSH  PSW  ; SAVE DATA
MVI  A,0FFH  ; NEXT CHARACTER IS DATA
CALL  HOSTOUT
POP  PSW
PUSH  PSW
CALL  HOSTOUT  ; SEND DATA
POP  PSW  ; RESTORE DATA
RET

* MDSDONE - SEND MDS DONE COMMAND *

MDSDONE MVI  A,55H  ; NEXT CHARACTER IS COMMAND
CALL  HOSTOUT
MVI  A,'Q'  ; QUIT COMMAND
CALL  HOSTOUT
RET

* MDSRDY - MDS IS READY FOR INPUT OR OTHER ACTION BY HOST *

MDSRDY MVI  A,00H  ; NEXT CHAR. IS READY FLAG
CALL  HOSTOUT
MVI  A,00H
CALL  HOSTOUT
*** DATA STORAGE AREAS - IN ONBOARD RAM ***

<table>
<thead>
<tr>
<th>ORG</th>
<th>RAM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSTREYFL ES</td>
<td>1</td>
<td>HOST READY FLAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = NOT READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OFFH = READY</td>
</tr>
<tr>
<td>MATCHAER EW</td>
<td>0</td>
<td>STORAGE FOR FIRST ADDRESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OF MATCH</td>
</tr>
<tr>
<td>LOCCOUNT DS</td>
<td>1</td>
<td>STORAGE FOR BYTE COUNT</td>
</tr>
<tr>
<td>START EW</td>
<td>0</td>
<td>STORAGE FOR START &amp; FINISH ADDRESSES</td>
</tr>
<tr>
<td>FINISH EW</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>OPTION ES</td>
<td>63</td>
<td>STORAGE FOR OPTION SELECTED 32 LEVEL STACK</td>
</tr>
<tr>
<td>STACK ES</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATABUFF DS</td>
<td>25</td>
<td>STORAGE FOR LOCATE SEQUENCE</td>
</tr>
</tbody>
</table>
APPENDIX E

MDS MEMORY TEST PROGRAM LISTING

****************************************************************************************************
*  MDS MEMORY DIAGNOSTIC
*  *  VERSION 2.5  11 MAY 1981
*  *  THIS PROGRAM IS A REVISION OF THE Z-80 MEMORY TEST
*  *  PROGRAM PUBLISHED IN THE FEBRUARY 1981 ISSUE OF
*  *  "DR. DOBB'S JOURNAL OF COMPUTER CALISTHENICS & ORTHODONTIA"
*  *  THE PROGRAM HAS BEEN TRANSLATED TO 8080 ASSEMBLY CODE AND
*  *  MODIFIED TO OPERATE ON THE ALTOS AND MDS SYSTEMS.
*  *  REVISIONS MADE BY LT. STEPHEN M. HUGHES FOR USE IN THESIS
*  *  AS STATED IN THE ORIGINAL TEXT, "FURTHER RESALE OF THIS
*  *  PROGRAM IS PROHIBITED", UNLESS INCLUDED IN THE BODY OF THE
*  *  REVISIONIST'S THESIS.
*  ****************************************************************************************************

ORG     4000H
USRIO   EQU     0005H ;USER I/O CALL
BESPACE EQU     08H  ;ASCII BACKSPACE
ESC     EQU     1BH   ;ASCII ESCAPE CODE
CR      EQU     0DH   ;ASCII CARRIAGE RETURN
LF      EQU     0AH   ;ASCII LINE FEED
RCNT    EQU     3     ;SEQUENTIAL READS
W_CNTL  EQU     3     ;SEQUENTIAL WRITES
MEM     DI
LXI     SP,STACK
LXI     B,TEND
LXI     H,MEMT1
CALL    CHA

* TEST STARTS HERE *

MEM01   CALL    CRLF ;MAKE OUTPUT PRETTY
LXI     H,0000H ;INITIALIZE PAS COUNT,
               ;CUMULATIVE ERROR COUNT
AND ADDRESS 'OR' PRODUCT

SHLD MEMF
SHLE MEMX
SHLD MEML
LXI H,-1
SHLD MEMK
LXI H,MEMA
CALL DSPLY

* GET TEST MODE *

MEM03 MVI A,1
STA MEMP
LXI H,MEMN
CALL DSPLY
CALL CRLF
MVI A,'>'
CALL USROUT
CALL USRIN
ORI 20H
CPI 'e'
JZ MEM04
CPI 'i'
JZ MEM05
CPI 't'
JNZ MEM03
XRA A
STA MEMP

* GET MEMORY TEST LIMITS *

MEM04 LXI H,MEMB
CALL DSPLY
CALL ENTR
MOV A,H
ORA A
JM MEM05
LXI D,TEND
PUSH H
MOV A,L
SUB E
MOV L,A
MOV A,H
SBB L
MOV H,A
POP H
JP MEM05

MEM045 LXI H,MEMT
CALL DSPLY

; AND ADDRESS 'OR' PRODUCT

;INIT. ADDRESS 'AND'

;PRINT PROGRAM TITLE

;SET DEFAULT = ITEMIZE

;PRINT SELECT I,T OR E

;PRINT SELECT I,T OR E

;PROVIDE A CUE MARK

;WAIT FOR INPUT

;MAKE LOWER CASE

;IF E, EXIT

;IF I, ITEMIZE ERRORS

;IF T, PRINT TOTAL ERRORS

;ONLY

;IF NONE, TRY AGAIN

;SET TOTAL ONLY FLAG

;PRINT ENTER FBA

;GET 16 BIT ADDRESS

;IF UPPER BYTE OF FBA IS

;NEGATIVE, OK TO USE

;SO JUMP

;OTHERWISE, MAKE SURE FBA

;IS NOT WITHIN TEST PROGRAM

;AREA

;(HL = HL - DE - C)

;FBA IS OK, JUMP

;IF FBA IS WITHIN TEST PROGRAM

;AREA, SET IT TO END OF
LXI H, TEND             ; PROGRAM & PRINT A WARNING
SHLD MEMI              ; SAVE FIRST BYTE ADDRESS (FBA)
LXI H, MEMC            ; PRINT ENTER LAST BYTE ADDRESS (LBA)
CALL DSPLY
CALL ENTR             ; ... ACCEPT ADDRESS
PUSH H                 ; SAVE LBA
PUSH H                 ; CLEAR CARRY FLAG
ORA A
PUSH H

LHLI MEMI
MOV L, H
MOV E, L
POP H
MOV A, L                ; MAKE SURE FBA < LBA
SUB E
MOV L, A
MOV A, H
SBB D
MOV B, A
JNC MEM06              ; IT'S OK, JUMP
POP H                  ; RESTORE STACK
POP B
LXI H, MEMU          ; FBA IS >= LBA SO PRINT
CALL DSPLY             ; ERROR MESSAGE
JMP MEM04              ; AND ACCEPT ADDRESSES AGAIN

* ALL ADDRESSES OK NOW *

MEM06
POP B                ; BC = LBA
LXI H, MEMG+5        ; CONVERT IT FOR PRINTING
CALL CHA
PUSH H               ; CONVERT FBA FOR PRINTING
LHLI MEMI            ; (BC = CONTENTS OF MEMI AND MEMI + 1)
MOV B, H
MOV C, L
POP H
LXI H, MEMG
CALL CHA
POP H

MEM08
LXI E, MEMV          ; PRINT ABORT INSTRUCTION
CALL DSPLY
POP E                ; LE = LBA
INK D                ; LBA = LBA + 1

* MAIN LOOP OF MEMORY TEST BEGINS HERE *
* BEGIN A PASS *

131
MEM1   MVI   C,1 ; INITIALIZE PATTERN NO.
LXI   3,0000H ; INITIALIZE ERROR COUNT
SHLD   MEME

* TEST ALL OF DESIGNATED MEMORY FOR CURRENT PATTERN *

* WRITE PATTERN INTO MEMORY *

MEM15  MVI   B,WCNT ; INIT. WRITES COUNTER
MEM2   LHLD   MEMI ; GET FIRST BYTE ADDRESS TO TEST
CALL   USRSTAT ; CHECK KEYBOARD
RRC   CC   MEM5 ; IF CHARACTER WAITING,
            ; INTERRUPT TEST
PUSH   B ; SAVE PATTERN AND WRITES COUNTER
MEM21  CALL   PATTN ; COMPUTE PATTERN FOR THIS MEMORY ADDRESS
            ; WRITE IT
MOV   M,A
INX   H ; ADVANCE MEMORY ADDRESS
MOV   A,L ; CHECK IF END OF AREA TO BE TESTED
CMP   E ; LOOP, NOT YET
JNZ   MEM21
MOV   A,H
CMP   D
JNZ   MEM21 ; LOOP, NOT DONE YET
POP   B ; GET WRITES COUNTER
LCR   B ; WRITE PATTERN OVER AND OVER
JNZ   MEM31
MVI   B,RCNT ; INIT. READS COUNTER

* NOW READ PATTERN BACK FROM MEMORY AND COMPARE TO COMPUTED PATTERN. IF DIFFERENCE IS FOUND ON FIRST READ, ASSUME A POSSIBLE WRITE ERROR. IF FIRST READ MATCHES, COMPARE 16 MORE TIMES LOOKING FOR SOFT READ ERRORS. *

MEM3   LHLD   MEMI ; GET FBA OF MEMORY TO TEST
CALL   USRSTAT ; CHECK KEYBOARD
ORA   A ; IF CHARACTER WAITING,
CNZ   MEM5 ; INTERRUPT TEST
PUSH   B ; SAVE PATTERN AND READS COUNTER
MEM31  CALL   PATTN ; COMPUTE PATTERN FOR THIS MEMORY ADDRESS
            ; SAVE IT
MOV   B,A
MOV   A,M ; READ MEMORY
CMP   B ; IS DATA CORRECT?
JZ   MEM32 ; YES, JUMP
MOV   M,B ; WRITE THE CORRECT DATA

132
CALL ERR1 ;DATA DOESN'T MATCH, PRINT POSSIBLE WRITE ERROR AUDIT
JMP MEM35 ;TEST NEXT ADDRESS DATA MATCHED ON FIRST TRY TRY FOR A SOFT READ ERROR
MEM32 SUB M ;BY HITTING THIS ADDRESS A
ADD M SOLID 16 TIMES
SUB M
ADD M
SUB M
ADD M
SUB M
ADD M
SUB M
ADD M
SUB M
ADD M
SUB M
ADD M
CMP B ;DOES DATA STILL MATCH?
CNZ ERR2 ;NO, PRINT POSSIBLE REAL ERROR AUDIT
MEM35 INX H ;ADVANCE MEMORY ADDRESS MOV A,L ;CHECK IF REACHED END OF MEMORY CMP E ;AREA TO BE TESTED JNZ MEM31 ;NOT DONE YET, LOOP MOV A,H CMP L JNZ MEM31 ;NOT DONE YET, LOOP POP B ;RESTORE PATTERN AND READ COUNTER DCR B ;READ PATTERN OVER AND OVER JNZ MEM3
* LONE WITH ONE PATTERN, ADVANCE TO NEXT AND CHECK FOR END OF PASS *
INR C ;INCREMENT PATTERN MOV A,C MOV CPI 11 ;DONE YET? JNZ MEM15 ;NO, LOOP JMP MEM6 ;AUDIT THIS PASS

* CHARACTER WAITING ON KEYBOARD, INTERRUPT TEST AND CHECK FOR EXIT REQUEST *
MEM5 CALL USRIN ;GET INPUT
CPI 04H ;D - FREEZE ACTION
JZ DISPSTP
ORI 20H ;FOLD TO LOWER CASE
CPI 'i'
JZ MAKEI
CPI 't'
JZ MAKET
CPI 'e'
JNZ STACKIT ;RESTART TEST IF NOT E
MEM55 LXI H,MEMM ;EXIT FROM TEST, PRINT GOODBYE
CALL DSPLY
DISPSTP CALL USRIN ;WAIT FOR ANY KEY TO RESUME
CALL BSOUT ;ACTION
RET
STACKIT LXI SP,STACK ;RESET STACK
JMP MEM01 ;RESTART TEST
MAKEI MVI A,1
STA MEMP
CALL BSOUT
RET
MAKET MVI A,0
STA MEMP
CALL BSOUT
RET

* DONE WITH PASS THROUGH MEMORY *
* PRINT CONSOLE AUDIT IN THE FORM:
* PASS: xxxx ERRORS: xxxx CUM. ERRORS: xxxx
* (IF CUMULATIVE ERRORS > ZERO THEN ALSO PRINT)
* AND: xxxx OR: xxxx *

MEM6 PUSH D ;SAVE LBA+1
PUSH H ; (BC = CONTENTS OF MEMF
LHLD MEMF ; AND MEMF + 1)
MOV B,H
MOV B,L
POP H
INX B ;COUNT PASSES
PUSH H ; (MOV BC TO MEMF)
MOV H,B
MOV L,C

134
SHLD MEMF
POP H
LXI 3, MEMG1
CALL CHA
PUSH H
; (BC = CONTENTS OF MEME
; AND MEME + 1)

LHLD MEME
MOV B, H
MOV C, L
POP H
LXI H, MEMG2
CALL CHA
PUSH H
; (BC = CONTENTS OF MEMX
; AND MEMX + 1)

LHLD MEMX
MOV B, H
MOV C, L
POP H
LHLD MEME
DAD B
; ACCUMULATE ERRORS FOR
; ALL PASSES

SHLD MEMX
PUSH H
; FORMAT CUMULATIVE ERRORS
POP E
LXI H, MEMG23
CALL CHA
MVI A, CR
; SET UP OUTPUT TO SKIP 'AND'
; & 'OR' OF FAILING MEMORY
; ADDRESSES IF NO ERRORS HAVE
; BEEN FOUND

STA MEMG25
LHLD MEMX
MOV A, H
ORA L
JZ MEM67
MVI A, ""
; REMOVE THE CARRIAGE RETURN
; FROM THE OUTPUT STRING

STA MEMG25
PUSH H
; (BC = CONTENTS OF MEMK
; AND MEMK + 1)

LHLD MEMK
MOV E, H
MOV C, L
POP H
LXI H, MEMG3
; CONVERT LOGICAL 'AND' OF
; FAILING ADDRESSES
CALL CHA
PUSH H
; (BC = CONTENTS OF MEML
; AND MEML + 1)

LHLD MEML
MOV B, H

135
MOV C,L
POP H
LXI H,MEMG4 ;CONVERT LOGICAL 'OR' OF
; FAILING ADDRESSES
CALL CHA
LXI H,MEMG ;PRINT PASS AUDIT
CALL DSPLY
LDA MEMJ ;ROTATE BIT CROSSTALK SC THAT
RLC ; OVER EIGHT PASSES ALL BIT
STA MEMJ ; PATTERNS WILL BE USED
POP D ;RESTORE LBA+1
JMP ME67 ;START ANOTHER PASS

* ERROR AUDITING ROUTINE *
* CONSOLE OUTPUT OF THE FORM: *
*   A=xxxx P=xx C-xx XOR=xx ERROR-TYPE *
*   A = FAILING ADDRESS
*   P = CALCULATED PATTERN
*   C = ACTUAL CONTENTS OF ADDRESS
*   XOR = EXCLUSIVE OR OF PATTERN AND CONTENTS
*      (ISOLATES FAILING BIT(S))
*   ERROR-TYPE = RD PRESUMED READ (SOFT) ERROR
*        WT PRESUMED WRITE (HARD) ERROR *

ERR1 PUSH PSW ;POSSIBLE WRITE ERROR
MVI A,"W"
STA MEMD5
MVI A,"T"
STA MEMD5+1
POP PSW
JMP ERROR

ERR2 PUSH PSW ;POSSIBLE READ ERROR
MVI A,"R"
STA MEMD5
MVI A,"L"
STA MEMD5+1
POP PSW

ERROR PUSH B ;SAVE ALL REGISTERS DURING
; ERROR AUDIT
PUSH D
PUSH H
PUSH PSW
XRA B ;LOGICAL EXCLUSIVE 'OR' OF
; CALCULATED PATTERN AND
; ACTUAL MEMORY CONTENTS
MOV C,A
LXI H, MEM4 ; CONVERT 'OR' FOR OUTPUT
CALL CHAB
POP PSW ; GET MEMORY CONTENTS AND
MOV C, A ; CONVERT IT FOR OUTPUT
LXI H, MEM3
CALL CHAB
MOV C, B ; CONVERT PATTERN
LXI H, MEM2
CALL CHAB
POP B ; CONVERT CURRENT MEMORY ADDRESS
PUSH B
LXI H, MEM1
CALL CHA
INX H ; COUNT ERRORS THIS PASS
SHLD MEME
INX H
SHLD MEME
POP D ; GET CURRENT MEMORY ADDRESS
PUSH D
LHLD MEMK
MOV A, L ; SAVE LOGICAL 'AND' OF
ANA H ; FAILING ADDRESSES
MOV H, A
MOV A, E
ANA L
MOV L, A
SHLL MEML
MOV A, L ; SAVE LOGICAL 'OR' OF
ORA H ; FAILING ADDRESSES
MOV H, A
MOV A, E
ORA L
MOV L, A
SHLD MEML
LDA MEMP ; CHECK ITEMIZE ERRORS FLAG
ORA A
JZ ERR9 ; SKIP PRINT IF FLAG = 0
LXI H, MEMD ; PRINT ERROR AUDIT
CALL DSPLY
ERR9 POP H ; RESTORE REGISTERS AND
POP D ; RETURN TO MAIN TEST
POP B
RET

* COMPUTE TEST DATA PATTERN FOR GIVEN MEMORY ADDRESS *
* CALL WITH HL = MEMORY ADDRESS *
C = PATTERN COUNTER

* RETURN A = DATA PATTERN *

PATTN PUSH H
MVI B,0
LXI H,PATT0-3
DAD B
DAD B
DAL B
XTHL
NOP
RET

PATT0 JMP PAT1 ;1 CAMBRIDGE PATTERN
JMP PAT2 ;2 ADDRESS
JMP PAT3 ;3 ALTERNATE 1's AND 0's
JMP PAT4 ;4 ADDRESS INVERSE
JMP PAT5 ;5 ALTERNATES 0's AND 1's
JMP PAT6 ;6 ALL ONES
JMP PAT7 ;7 CAMBRIDGE INVERSE
JMP PAT8 ;8 ALL ZEROS
JMP PAT9 ;9 BIT CROSSTALK
JMP PAT10 ;10 BIT CROSSTALK INVERSE

PAT1 MOV A,L ;CAMBRIDGE PATTERN
RRC
RRC
RRC
XRA H
ANI 1
JZ ONES

ZEROS XRA A
RET
ONES MVI A,0FFH
RET

PAT2 MOV A,L ;ADDRESS
RET

PAT3 MVI A,0AAH ;ALTERNATE 1's AND 0's
RET

PAT4 MOV A,L ;ADDRESS INVERSE
CMA
RET

PAT5 MVI A,55H ;ALTERNATE 0's AND 1's
RET

PAT6 EQU ONES ;ALL BITS = ONE
PAT7  MOV  A,L ;CAMBRIDGE INVERSE
    RRC
    RRC
    RRC
    XRA  H
    ANI  1
    JZ  ZEROS
    JMP  ONES

PAT8  EQU  ZEROS ;ALL BITS = ZERO

PAT9  MOV  A,L ;BIT CROSSTALK
    RAR
    JC  PAT91
    LDA  MEMJ
    RET

PAT91 LDA  MEMJ
    CMA
    RET

PAT10 MOV  A,L ;BIT CROSSTALK INVERSE
    RAR
    JNC  PAT91
    LDA  MEMJ
    RET

*  BINARY TO HEX ASCII CONVERSION, 16 BITS *
*  CALL  HL = ADDRESS FOR 4 CHAR ASCII OUTPUT STRING
*  BC = 16 BIT BINARY DATA
*  RETURNS HL,DE,BC UNCHANGED
*     A  =  GARBAGE *

CHA  PUSH  H ;SAVE REGISTERS
    PUSH  D
    PUSH  E
    INX  H
    INX  H
    INX  H
    MVI  D,4 ;CHAR COUNTER

CHA1  MOV  A,C ;NEXT 4 BITS
    ANI  0FH
    CPI  0AH ;IS IT A-F?
    JC  CHA15 ;NO
    ALI  7 ;YES

CHA15 ADI  "0" ;FORM ASCII
    MOV  M,A ;STORE THIS CHARACTER
    LDS  B ;BACK UP THROUGH OUTPUT AREA
    MVI  E,4 ;DOUBLE RIGHT
ORA A ;SHIFT 4 BITS

CHA2
MOV A,B
RAR
MOV B,A
MOV A,C
RAR
MOV C,A
ICR E ;DECREMENT SHIFT COUNTER
JNZ CHA2 ;STILL SHIFTING
DCR D ;DECREMENT CHARACTER COUNTER
JNZ CHA1 ;STILL CONVERTING
POP B ;RESTORE REGISTERS
POP D ;AND EXIT
POP H
RET

* BINARY TO HEX ASCII CONVERSION, 8 BITS *

* CALL HL = ADDRESS FOR 2 CHARACTER OUTPUT STRING
* C = 8 BIT BINARY DATA
*
* RETURN HL,EE,BC UNCHANGED
* A DESTROYED *

CHA1 PUS H ;SAVE REGISTERS
PUS D
PUS B
INX H
MVI D,2
JMP CHA2

* PRINT CHARACTER STRING *

* CALL HL = FIRST BYTE ADDRESS OF OUTPUT STRING
* (MUST END WITH ASCII CARRIAGE RETURN) *

DSPLY CALL CRLF
LSPLY1 MOV A,M
CALL USROUT ;OUTPUT THIS CHARACTER
CPI CR ;END OF STRING?
RZ ;YES, EXIT
INX H ;NO, BUMP STRING POINTER
JMP DSPLY1

* GET KEYBOARD ENTRY OF HEX INTEGER *

* RETURN HL = 16 BIT BINARY DATA *
ENTR  LXI  H,0000H  ;INITIALIZE DATA
CALL  CRLF  ;SEND CARRIAGE RETURN &
        ;LINE FEED
MVI   A,'>'  ;SEND A CUE MARK
CALL  USROUT
MVI   C,4  ;CHAR. COUNTER
ENTR1 CALL  USRIN  ;GET 1 CHARACTER
CPI   CR  ;CARRIAGE RETURN?
        ;YES, EXIT
        ;LINE FEED?
        ;YES, EXIT
RZ    CPI   'A'  ;IS IT 0-9?
JC    ENTR15  ;YES
ANI   OFH  ;NO, FORCE LOWER CASE
ENTR15 DAD  H  ;SHIFT PREVIOUS DATA LEFT
        ;4 BITS
LAC  EAC  H
DAD  H
JC    ENTR3  ;IF OVERFLOW, PRINT '?
CPI   '0'  ;IS IT 0-F?
JC    ENTR3  ;Illegal CHARACTER
CPI   'F'+1
JNC   ENTR3  ;Illegal CHARACTER
CPI   'A'  ;IS IT A-F?
JC    ENTR3  ;NO, IT'S 0-9
ADI   9  ;ADD FUDGE FACTOR
ENTR2 ANI   OFH  ;ISOLATE 4 BITS
ORA   L  ;MERGE WITH PREVIOUS DATA
MOV   L,A
DCR   C  ;COUNT CHARACTERS
RZ    JMP   ENTR1  ;EXIT IF 4 RECEIVED
ENTR3 MVI   A,'?'
CALL  USROUT  ;GET ANOTHER CHARACTER
JMP   ENTR  ;PRINT QUESTION MARK
        ;AND RESTART ENTRY

* PRINT CARRIAGE RETURN AND LINE FEED *

CRLF MVI   A,CR
CALL  USROUT
MVI   A,LF
CALL  USROUT
RET

* MISCELLANEOUS MESSAGES AND DATA AREA *

MEMA  DB   '8080 MEMORY TEST - VERSION 2.5',LF,CR
MEMB  LB   'ENTER ADDRESS OF FIRST MEMORY BYTE'
        'TO TEST:',CR
MEMC DB 'ENTER ADDRESS OF LAST MEMORY BYTE',CR
MEMD DB 'ADDRESS=',
MEML1 DB '$$$$ PATTERN=',
MEML2 DB '$$ CONTENTS=',
MEML3 DB '$$ XOR=',
MEML4 DB '$$ TYPE=',
MEML5 DB ',CR,
MEME DW 0,CR ;ERRORS THIS PASS
MEMF LW 0 ;PASS COUNT
MEMG DB '$$$$$$$$ PASS:',
MEMG1 DB '$$$$ ERRORS:',
MEMG2 DB '$$$$ CUM. ERRORS:',
MEMG25 DB CR ' AND:',
MEMG3 DB '$$$$ OR:',
MEMG4 DB '$$$$',CR
MEMI DW 0 ;FIRST BYTE ADDRESS TO TEST
MEMJ LB 0FEH ;BIT CROSSTALK PATTERN
MEMK LW -1 ;LOGICAL 'AND' OF FAILING
MEML LW 0 ;LOGICAL 'OR' OF FAILING
MEMM DB LF 'GOODBYE',CR
MEMN DB 'I=ITEMIZE ERRORS, ','T=PRINT ERROR TOTAL ONLY, ','E=EXIT TEST ',CR
MEMP LB 0 ;FLAG 1=ITEMIZE, 0=TOTAL
MEMT LB 'END OF PROGRAM USED AS FIRST '
MEMT1 DB 'ADDRESS TO TEST = ','
MEMU DB 'ERROR: LAST BYTE ADDRESS LESS '
MEMV LB LF ' THAN FIRST BYTE ADDRESS.',CR
MEMW DB LF 'TO ABORT TEST PUSH ANY KEY'
MEMX LW 0 ; CUMULATIVE ERROR COUNT

USRIN PUSH B ;GET INPUT FROM HOST CONSOLE
CALL USRIO
POP H
POP B
RET

USROUT PUSH B ;SEND CHARACTER TO HOST
PUSH D ;CONSOLE
PUSH H
MVI C, 2
CALL USRIO
POP H
POP D
POP B
RET

USRSTAT
PUSH B
PUSH C
PUSH H
MVI C, 3
CALL USRIO
POP H
POP D
POP B
RET

BSOUT
MVI A, BKSPACE
CALL USROUT
RET

STACK
DS 64
TEND EQU $+2
END 100H
APPENDIX F
SAMPLE MENU LISTING

<table>
<thead>
<tr>
<th>HOST COMMANDS</th>
<th>MDS COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. SUPPRESS PRINTING MENU</td>
<td>G. DOWNLOAD HEX FILE — DISK TO MDS MEMORY</td>
</tr>
<tr>
<td>B. DO NOT SUPPRESS PRINTING MENU</td>
<td>H. UPLOAD MDS MEMORY TO HEX DISK FILE</td>
</tr>
<tr>
<td>C. BASIC INSTRUCTIONS</td>
<td>I. EXAMINE/SET MDS MEMORY LOCATION(S)</td>
</tr>
<tr>
<td>D. HEXADECIMAL ADD &amp; SUBTRACT</td>
<td>J. CONTINUOUS SET OF MDS MEMORY</td>
</tr>
<tr>
<td>E. RETURN SYSTEM CONTROL TO HOST</td>
<td>K. FILL MDS MEMORY WITH SPECIFIED BYTE</td>
</tr>
<tr>
<td>F. RETURN TO CP/M</td>
<td>L. LOCATE BYTE SEQUENCE IN MDS MEMORY</td>
</tr>
<tr>
<td></td>
<td>M. DUMP MDS MEMORY LOCATION(S) TO CONSOLE</td>
</tr>
<tr>
<td></td>
<td>N. EXECUTE MDS MEMORY FROM SPECIFIED LOCATION</td>
</tr>
</tbody>
</table>

SYSTEM STATUS: HOST IN CONTROL; NO MENU SUPPRESSION

INPUT MENU OPTION >
APPENDIX G

SAMPLE BASIC INSTRUCTION LISTING

BASIC AMES INSTRUCTIONS:

A. THE PROMPT FOR INPUT OF DATA IS ">".
B. ALL INPUTS MAY BE IN UPPER OR LOWER CASE.
C. ADDRESS AND DATA INPUTS ARE EXPECTED TO BE IN HEX NOTATION.
D. TERMINATE INPUTS WITH A CARRIAGE RETURN OR LINE FEED.
E. NORMAL LINE EDITING ON INPUT IS AS IN CP/M AND MP/M.
F. FOR ADDRESS INPUTS, THE PROGRAM WILL ALWAYS TAKE THE LAST FOUR OR LESS HEX CHARACTERS ENTERED; FOR DATA INPUTS, THE LAST TWO OR LESS.
G. SOURCES OF COMMON ERROR ARE INVALID HEX DIGITS, TOO MANY OR TOO FEW DELIMITERS, AND ILLEGAL SYNTAX.
H. IN GENERAL, THE SAME DATA I/O FORMAT AS USED IN DIGITAL RESEARCH'S IIT IS USED HERE. FOR EXCEPTIONS, CONSULT THE USER'S MANUAL.
I. A QUESTION MARK ENTERED AFTER THE PROMPT WILL CAUSE THE INPUT FORMATS TO BE DISPLAYED.
J. IF THE ESCAPE KEY IS ENTERED DURING INPUT THEN THE USER IS RETURNED TO THE MENU.
K. FOR FURTHER DETAILS, CONSULT THE USER'S MANUAL

PRESS ANY KEY TO CONTINUE >
APPENDIX H

SAMPLE INPUT PARAMETER FORMAT LISTING

INPUT PARAMETER FORMATS ARE AS FOLLOWS:

MENU  >X  X IS OPTION SELECTION (A-N)
HEXARITH  >XXXX YYYY  YYYY & YYYY ARE HEX INTEGERS
DOWNLOAD  >FILENAME(.HEX)  (.HEX) IS OPTIONAL
UPLOAD  >FILENAME(.HEX)
        >XXXX YYYY  YYYY & YYYY ARE MDS HEX START AND END ADDRESSES FOR UPLOAD
EXAMINE MDS  >XXXX  XXXX IS FIRST MDS HEX ADDRESS TO EXAMINE AND SET
        >XXXX YYYY ZZ  XXXX IS HEX ADDRESS, YYYY IS HEX DATA
        AT THAT ADDRESS, ZZ IS CARRIAGE RETURN
        or ZZ IS NEW HEX DATA
        or ZZ IS ""
CONTINUOUS  >XXXX  XXXX IS MDS HEX START ADDRESS FOR FIRST CHANGE
        >AA BB CC .....  ARE HEX DATA FOR ENTRY INTO MDS MEMORY
        (255 ENTRIES MAX, INCLUDING DELIMITERS)
        IF ONLY A "" IS TYPED AFTER THE PROMPT, THE OPTION IS ENDED
FILL  >XXXX YYYY ZZ  YYYY & YYYY ARE MDS HEX START AND END ADDRESSES TO FILL BETWEEN;
        ZZ IS HEX DATA TO USE FOR FILL

PRESS ANY KEY TO CONTINUE >
LOCATE SEQ.  >XXXX( YYYY)  XXXX & YYYY ARE MIP HEX START AND
     >AA BB ... PP  OPTIONAL END ADDRESSES TO SEARCH BETWEEN
     ARE UP TO A 16 BYTE HEX SEQUENCE
DUMP     >XXXX( YYYY)  XXXX & YYYY ARE MIP HEX START AND
     OPTIONAL END ADDRESSES TO DUMP BETWEEN
EXECUTE  >XXXX  XXXX IS MIP HEX ADDRESS WHERE EXECUTION
           IS TO BEGIN

PRESS ANY KEY TO CONTINUE >
BIBLIOGRAPHY


DIGITAL RESEARCH CORPORATION, CP/M and MP/M Users Manuals, 1980.


## INITIAL DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td></td>
</tr>
</tbody>
</table>

1. Defense Technical Information Center  
   Cameron Station  
   Alexandria, Virginia 22314  
   2

2. Library, Code 0142  
   Naval Postgraduate School  
   Monterey, California 93940  
   2

3. Department Chairman, Code 62  
   Department of Electrical Engineering  
   Naval Postgraduate School  
   Monterey, California 93940  
   2

4. Associate Professor M. L. Cotton, Code 62Co  
   Department of Electrical Engineering  
   Naval Postgraduate School  
   Monterey, California 93942  
   2

5. Professor R. Panholzer, Code 62Pz  
   Department of Electrical Engineering  
   Naval Postgraduate School  
   Monterey, California 93940  
   1

6. LT Stephen M. Hughes, USN  
   1416 Sir Richard Road  
   Virginia Beach, Virginia 23455  
   1