

AD-A104 624

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/G 9/2

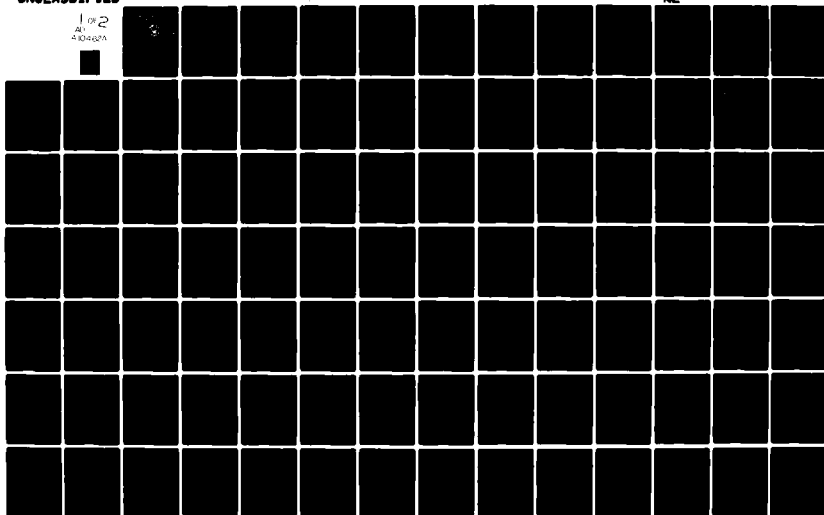
A MICROPROCESSOR DEVELOPMENT SYSTEM FOR THE ALTOS SERIES MICROC--ETC(U)

JUN 81 S M HUGHES

UNCLASSIFIED

ML

1 OF 2
AD-A104 624



AD A104624

② LEVEL II

NAVAL POSTGRADUATE SCHOOL
Monterey, California



DTIC
ELECTE
SEP 28 1981
B

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

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A104 624	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Microprocessor Development System for the ALTOS Series Microcomputers.		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis, June 1981
7. AUTHOR(s) Stephen Michael/Hughes		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE June 1981
		13. NUMBER OF PAGES 149
		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Microprocessor Development System ALTOS Microcomputer PRO-LOG STD bus CP/M, MP/M		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) An ALTOS series microcomputer is being used as the host computer in a micro-processor development system (MDS). The MDS hardware, consisting of the PRO-LOG STD bus, a 280 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.		

Approved for public release; distribution unlimited

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 M. Hughes

Approved by:

Mitchell S. Cotton

Thesis Advisor

R. Panholzer

by R.D. Strum
Second Reader

Acting

R.D. Strum

Chairman, Department of Electrical Engineering

William M. Loller

Dean of Science and Engineering

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.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability	
Dist	
A	

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 MDS 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: AMDS USERS GUIDE -----	39
	APPENDIX B: HOST AND MDS FLOW CHARTS FOR USER OPTIONS -	61
	APPENDIX C: AMDS HOST CONTROL SOFTWARE LISTING -----	71
	APPENDIX D: MDS 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 8002 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 8002.

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.

<u>PIN</u>	<u>MNEMONIC</u>	<u>DESCRIPTION</u>
1	+5VDC	Logic Power
2	+5VDC	Logic Power
3	GND	Logic Ground
4	GND	Logic Ground
5	VBB#1	Logic Bias #1 (-5V)
6	VBB#2	Logic Bias #2 (-5V)
7	D3	Data Bit 3
8	D7	Data Bit 7
9	D2	Data Bit 2
10	D6	Data Bit 6
11	D1	Data Bit 1
12	D5	Data Bit 5
13	D0	Data Bit 0
14	D4	Data Bit 4
15	A7	Address Line 7
16	A15	Address Line 15
17	A6	Address Line 6
18	A14	Address Line 14
19	A5	Address Line 5
20	A13	Address Line 13
21	A4	Address Line 4
22	A12	Address Line 12
23	A3	Address Line 3
24	A11	Address Line 11
25	A2	Address Line 2
26	A10	Address Line 10
27	A1	Address Line 1
28	A9	Address Line 9
29	A0	Address Line 0
30	A8	Address Line 8
31	WR*	Write to Memory or I/O
32	RD*	Read Memory or I/O
33	IORQ*	I/O Address Select
34	MEMRO*	Memory Address Select
35	IOEXP	I/O Expansion
36	MEMEX	Memory Expansion
37	REFRESH*	Refresh Timing
38	MCSYNC*	CPU Machine Cycle Sync.
39	STATUS 1*	CPU Status
40	STATUS 0*	CPU Status

Figure 1 - PRO-LOG STD Bus Pin Definitions

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

*Low-level active indicator

Figure 1 (cont'd)

B. SOFTWARE CONSIDERATIONS

The editing, assembling and debugging software needs for the AMDS were easily fulfilled 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 AMIS.

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

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 MDS 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 MDS 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 MDS 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 MDS 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 on 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.

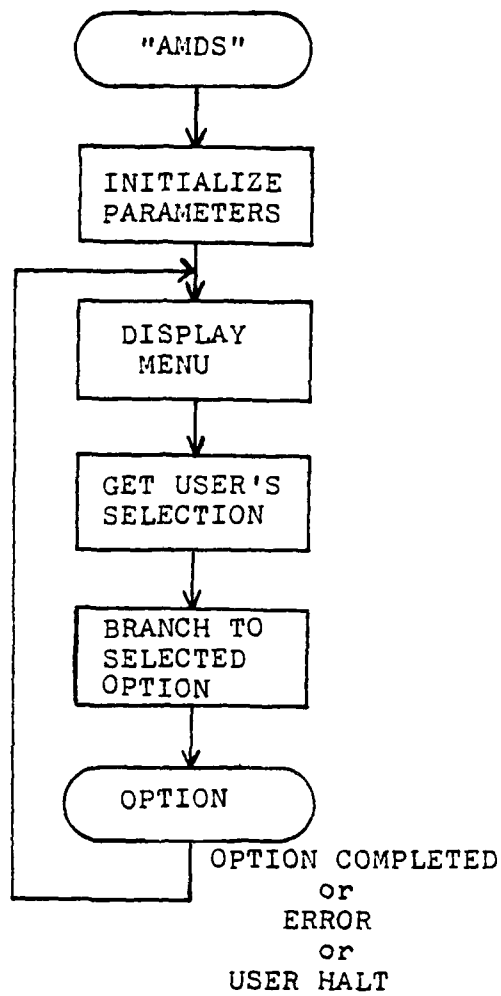


Figure 2 - HOST Control Program Flowchart

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 (DDT). 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 MDS. Because the MDS 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 9600 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 gleaned 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 STD 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

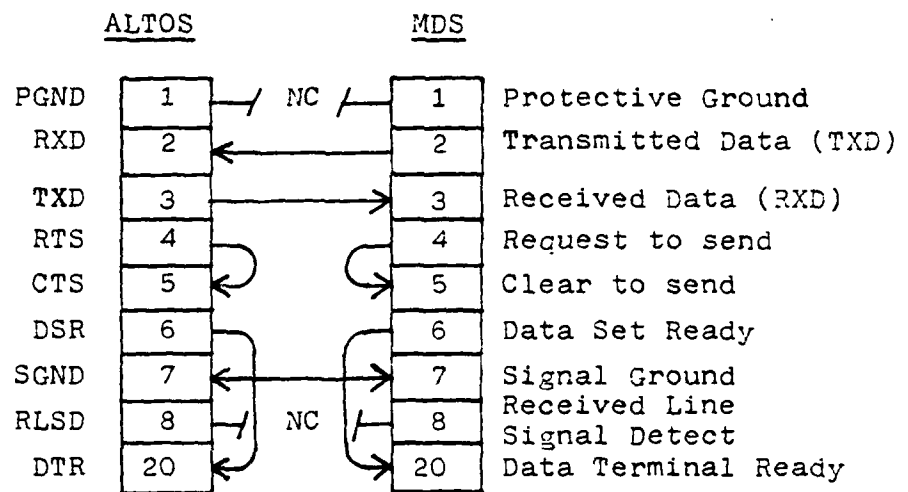
Implementation of the HCST control program is simply a matter of loading and executing the program via the normal CP/M method of typing in the name of the object file, in this case 'AMDS', followed by a carriage return or line feed.

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 MDS 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 ALTOS multi-user system and the MDS 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.



* 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 MDS and the use of different serial interfaces.

At present the PRO-LOG Corporation STD bus supports the 8080, 8085, Z80, Z80A and the 6800 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 6800 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, MDSIN, 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, O 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 NHSTCMD, 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 TEI text editor used widely at the Naval Postgraduate School. For this reason, the source code is broken into two files: AMDSP1.ASM containing the primary option routines, and AMDSP2.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=AMDSP1.ASM,AMDSP2.ASM
```

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

MDS 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 MDS 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 6800 and 280, 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 MDS 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 MDS, such '.COM' files cannot be downloaded and executed in MDS 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	RL	LA	RT	DATA	CK
----	----	----	----	------	----

- 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 0BFFFFH, 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 '.HEX' 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 usefulness 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 AMES 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

AMES 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

AMDS USERS GUIDE

1. INTRODUCTION

The ALTOS Microprocessor Development System (AMDS) 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 USART 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 AMDS.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 AMDS system is available in the student thesis by LT. Stephen M. Hughes, USN, titled "A Microprocessor Development System for the ALTOS Series Microcomputers".

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, TED, 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 MDS 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 AMIS concept is thus seen as less cumbersome than initially thought and actually allows for a very flexible working environment. The use of the AMIS 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 MDS 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 MDS reset switch is pressed. The MDS is now ready for use.

Next, the AMDS HOST control software is initiated from the ALTOS system console by typing 'AMDS', 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 DIFF = 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 DDT **)

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 MDS 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 MDS -

Option 'G' allows the user to download an INTEL Hex format file from disk to MDS 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 MDS 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 MDS 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 'D':

>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: >0BC3 . The system then fetches the data from that location and displays it as:

0BC3 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 EDT **)

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 LIT **)

L. LOCATE BYTE SEQUENCE IN MDS MEMORY -

Option 'L' allows the user to search MDS 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 0FFFFFFF. 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 MDS memory is displayed. If not found, an appropriate message is also displayed.

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

Option 'M' provides for a hexadecimal and ASCII MDS 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 DDT **)

If the user wishes to continue the dump after the initial dump completes, he may type in the letter 'D' to

dump the next 256 byte block. Any other input will return the user to the menu.

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

N. EXECUTE MDS MEMORY FROM A SPECIFIED LOCATION -

Option 'N' allows the user to pass system control to the MDS and let it execute a program in its memory. User input required is the MDS 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 MDS 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 MDS returns control to the HOST; when the 'E' option is selected; or when the MDS system is manually reset.

** For further discussion on the proper use of this option, see the section on 'TIPS FOR MDS 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 alphabetics.
- 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 RUBOUT 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/O 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 = 0FFH 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 0FFH

** 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 *)

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 MDS 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 MDS. (* 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 *)

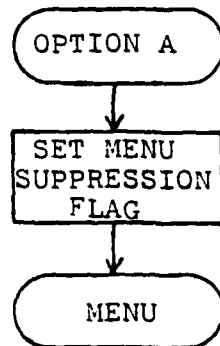
DISK READ ERROR - an attempt was made to read a disk file but was unsuccessful; check diskette media then the disk drive. (* this error returns the user to the menu *)

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

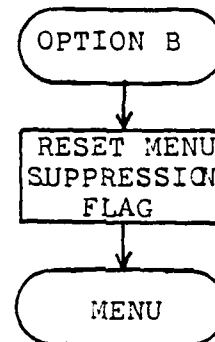
OUT OF DIRECTORY OR DISK STORAGE SPACE - ran out of space in one of these areas while attempting to write data to a disk; *** when this occurs, the data already written is deleted, i.e. NO PARTIAL files are saved ***. (* this error returns the user to the menu *)

APPENDIX B

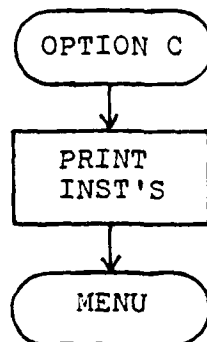
FLOWCHARTS FOR HOST AND MDS USER OPTIONS



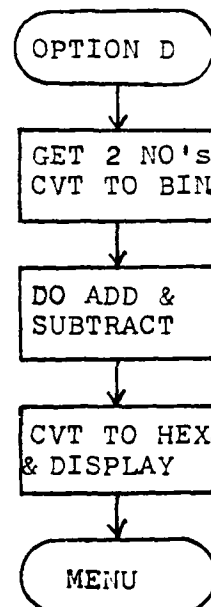
MENU SUPPRESSION



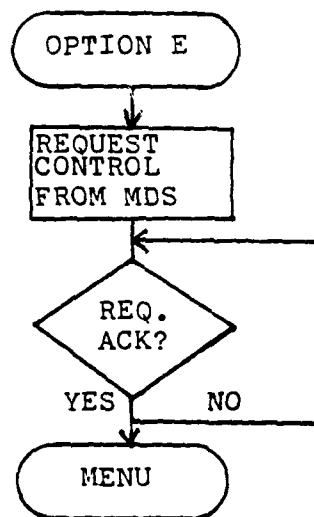
NO MENU SUPPRESSION



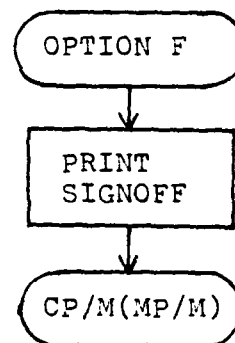
BASIC INSTRUCTIONS



HEX ADD/SUBTRACT

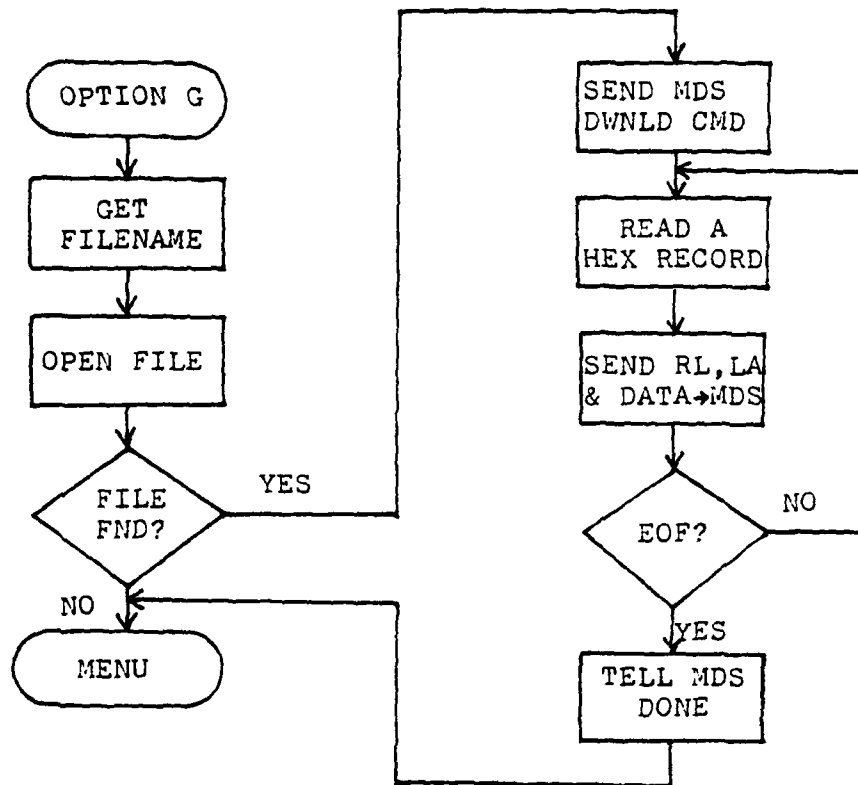


RETURN CONTROL TO HOST

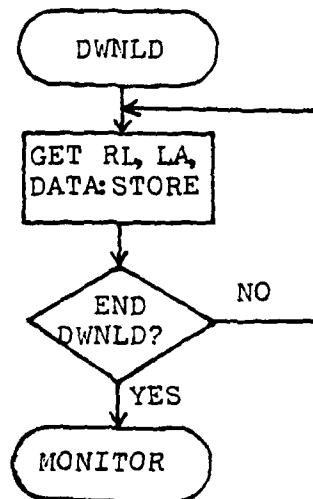


RETURN TO CP/M

DOWNLOAD HEX FILE TO MDS MEMORY

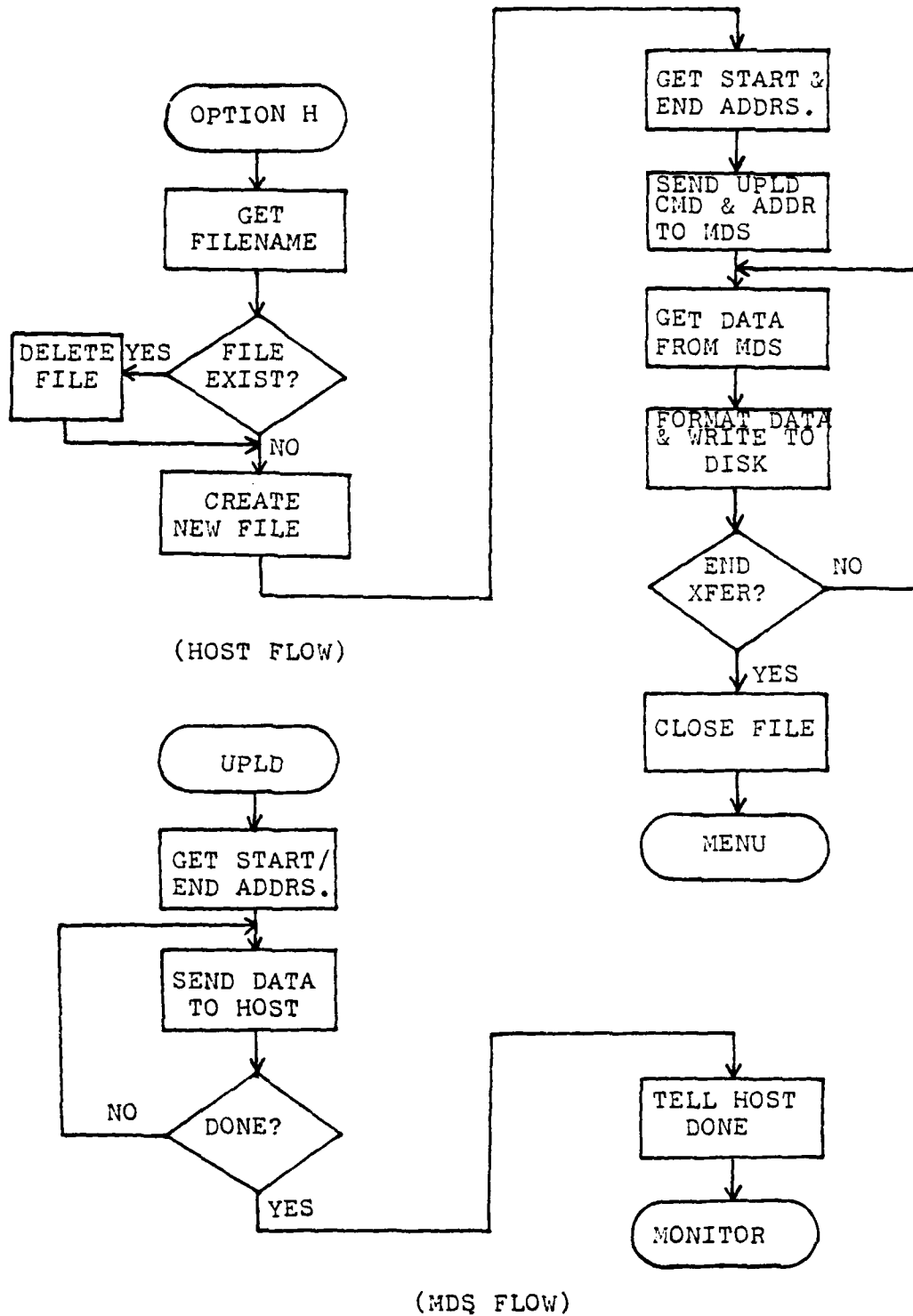


(HOST FLOW)

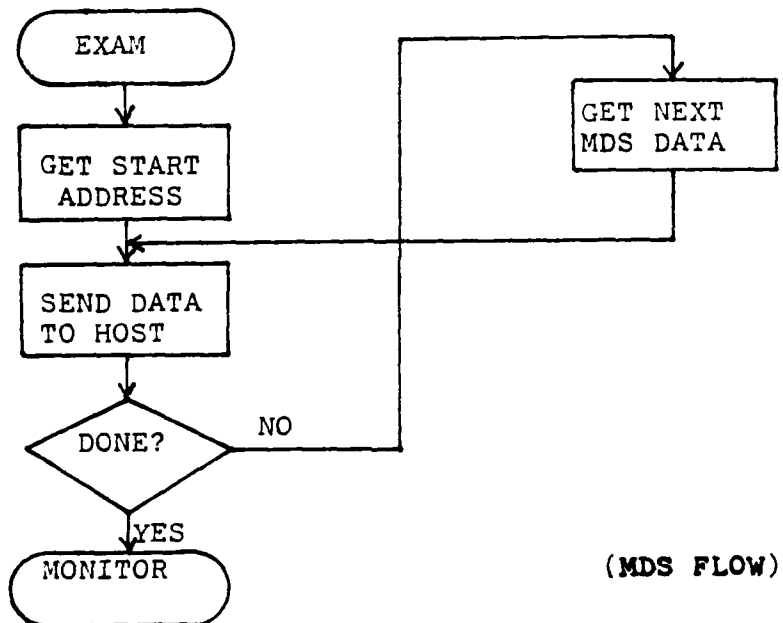
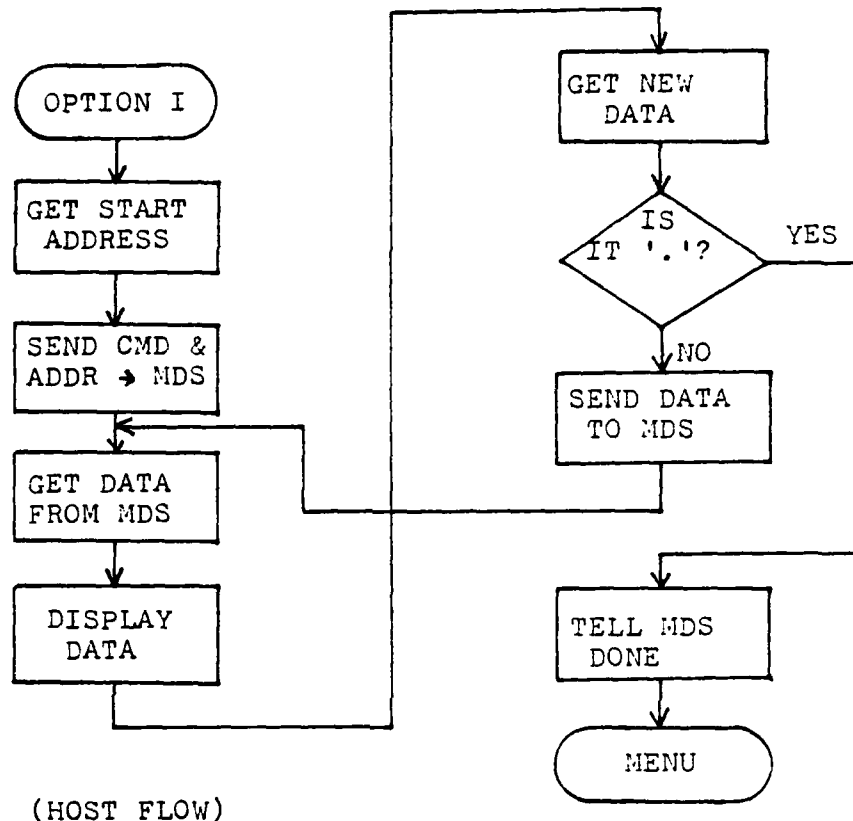


(MDS FLOW)

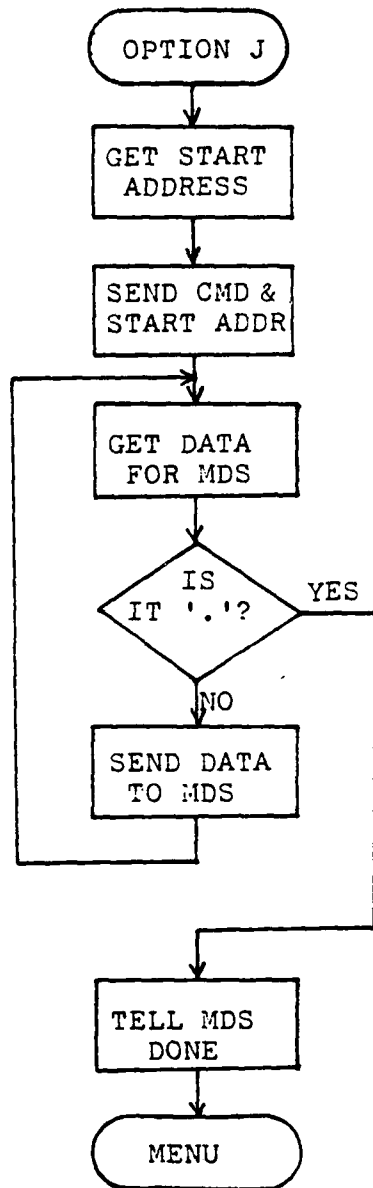
UPLOAD FROM MDS MEMORY TO HEX DISK FILE



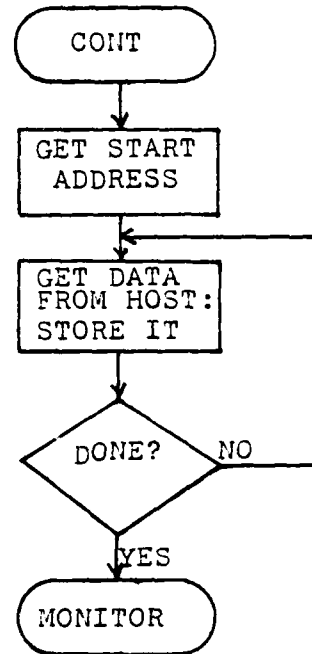
EXAMINE/SET MDS MEMORY



CONTINUOUS MDS MEMORY SET

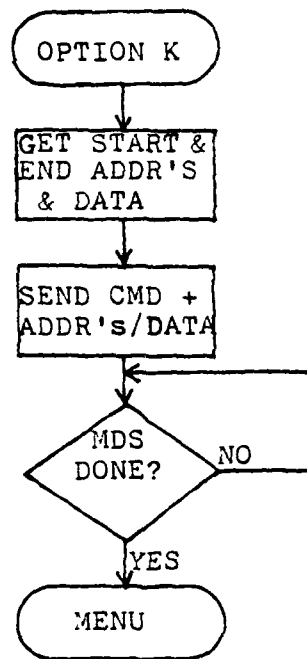


(HOST FLOW)

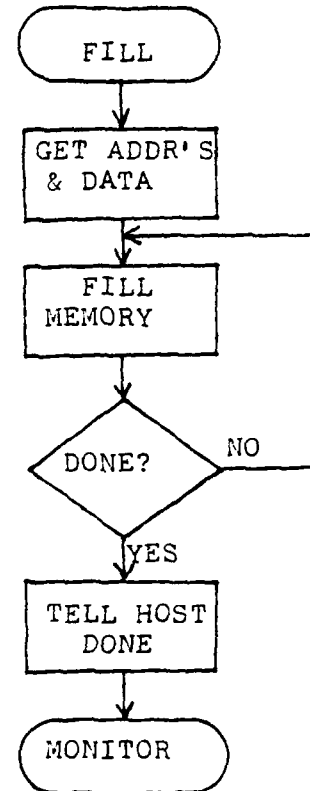


(MDS FLOW)

FILL MDS MEMORY WITH SPECIFIED BYTE

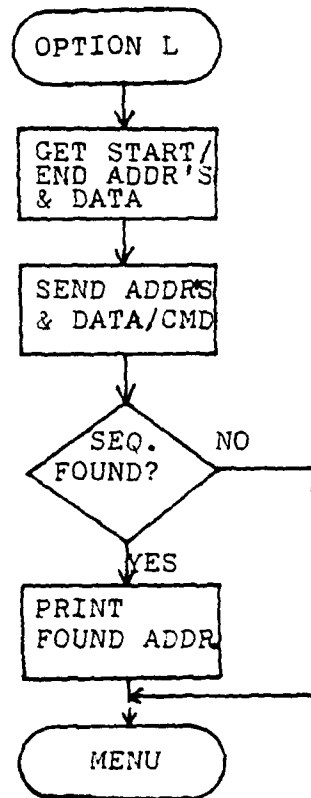


(HOST FLOW)

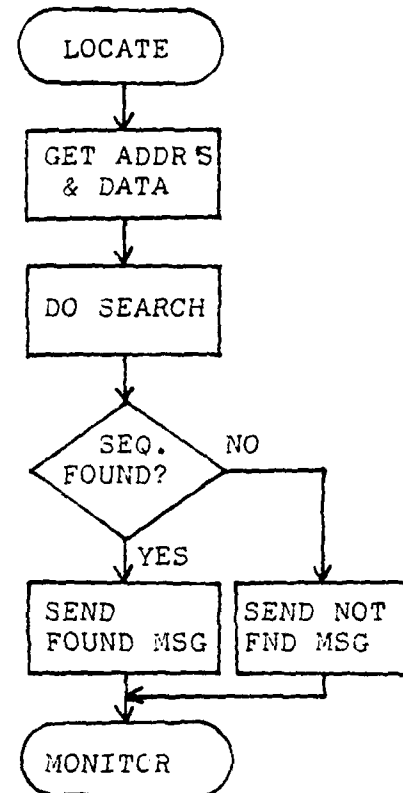


(MDS FLOW)

LOCATE BYTE SEQUENCE IN MDS MEMORY

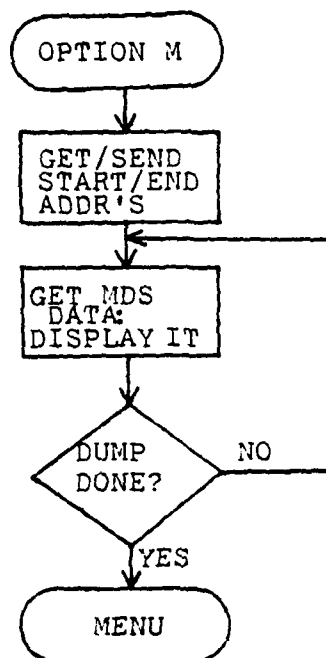


(HOST FLOW)

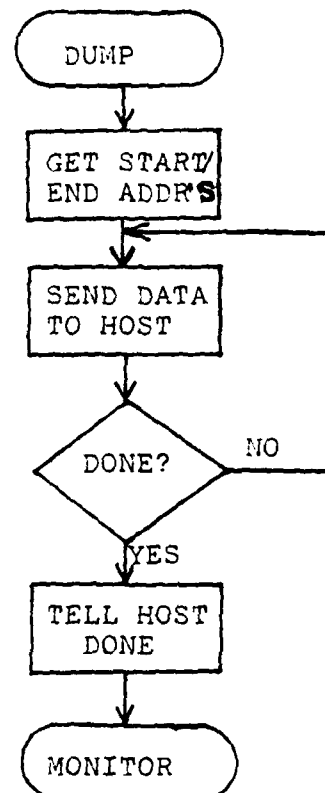


(MDS FLOW)

DUMP MDS MEMORY TO THE HOST CONSOLE

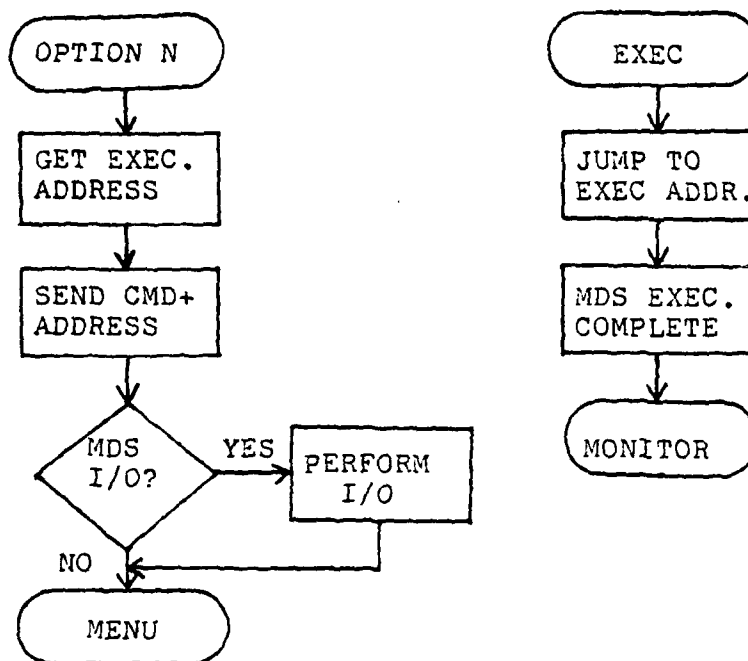


(HOST FLOW)



(MDS FLOW)

EXECUTE USER PROGRAM IN MDS MEMORY



(HOST FLOW)

(MDS FLOW)

APPENDIX C

AMDS 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 MES 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
BIOS     EQU      0005H      ;DOS ENTRY POINT
MSTATPT EQU      29H        ;MDS SIO STATUS PORT
MDATAPT EQU      28H        ;MDS SIO DATA PORT
CONIN    EQU      1         ;CONSOLE INPUT FUNCTION
CONOUT   EQU      2         ;CONSOLE OUTPUT FUNCTION
PRTSTRG  EQU      9         ;PRINT STRING TO CONSOLE
REALCON  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
CURRNTD  EQU      25        ;GET CURRENT DISK FUNCTION
SETDMA   EQU      26        ;SET IMA 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
BKSPCE   EQU      09H        ;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
EOF	EQU	1AH	;CONTROL Z - END OF FILE or ; BUFFER INDICATOR
NHSTCMD	EQU	6	;CURRENT NUMBER OF HOST CMLS
STACK	EQU	\$	;64 LEVEL STACK AVAILABLE
STARTER	XRA	A	;INITIALIZE HOST IN CONTROL.
	STA	SYSSTAT	
	STA	MENUSUPF	;MENU NOT SUPPRESSED
	LXI	D,SIGNON	;PRINT SIGNON AND BASIC ; INSTRUCTIONS
	CALL	PRINT	
MENU	XRA	A	;INIT. MDSRDYF EVERY TIME
	STA	MDSRDYF	
	INR	A	;DEFAULT TO NO MENU
	STA	MENUFLG	; SUPPRESSION ON MENU ERRORS ; OTHER THAN INVALID CHOICE
	MVI	A,48	;INIT. CONSOLE READ BUFFER
	STA	CONBUFF	; TO 48 CHARACTERS MAX
	LXI	SP,STACK	;SET STACK PCINTER
	LDA	MENUSUPF	;PRINT MENU?
	ORA	A	
	JNZ	MENU01	;NO
	LXI	D,MENUMSG	;YES
	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,MFDELERR	;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
	DCR	B	; AT END OF BUFFER YET?
	JNZ	MENU011	; NO, TRY AGAIN
	LDAX	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
        DAD      B
        DAD      B
        PCHL
        NOP
        NOP                   ;CHOICE VECTOR IS IN PC

```

* 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         ;CONTINOUS 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    D,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 *
* NUMBERS *

HEXARITH LXI    D,HEXMSG    ;PRINT VERIFICATION MESSAGE
        CALL   PRINT
        CALL   BUFFRD      ;GET INPUT
        MVI    A,1         ;ONE DELIMITER REQUIRED
        CALL   SCAN        ;CHECK FOR IT
        JNC    HEX1        ; ALL DELIMITERS OK
        MVI    A,2         ;DELIMITER ERROR
        JMP    ERROR
HEX1     CALL   GET4BIN      ;GET FIRST NUMBER
        SHLD   FIRST       ;SAVE IT
        CALL   GET4BIN      ;GET SECOND NUMBER
        SHLD   SECONL      ;SAVE IT
        MOV    B,H         ;BC = SECONL NUMBER
        MOV    C,L
        LHLD   FIRST       ;HL = FIRST NUMBER
        DAD    B           ;HL = HL + BC
        SHLD   SUM         ;SAVE SUM
        LHLD   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      H,HEXMSG2+7      ;CONVERT FOR PRINTING
CALL     CNVT16
LHLE     SUM              ;NOW PREPARE SUM FOR
PUSH     H                ; PRINTING
POP      B                ;BC = SUM
LXI      H,HEXMSG1+6
CALL     CNVT16
LXI      D,HEXMSG1        ;PRINT SUM & DIFFERENCE
CALL     PRINT
CALL     CRLF
JMP      MENU             ;RETURN TO MENU

```

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

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

```

DWNLD    LXI      D,DWNLDMSG      ;PRINT VERIFICATION MESSAGE
          CALL     PRINT
          CALL     GETFILEN       ;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 DOWNLOAD CMD TO MDS
          CALL     MDESCMD
          XRA      A             ;RESET CONTINUATION &
          STA      CONTF LG      ; FIRST THROUGH LCOP FLAGS
          STA      FIRSTIME
RDFILE    LXI      H,DSKBUF      ;POINTER TO DISK BUFFER
          CALL     READSK        ;READ IN AS MUCH AS POSSIBLE
          LXI      H,DSKBUF      ;NOW CONVERT IT TO BINARY &
          ; SEND IT TO MDS
RECHD     MOV      A,M           ;FIND ':' AS RECORD START
          CPI      ':'
          JZ      RECLN         ;FOUND IT
          INX      H
          CALL     EOFCK        ;END OF FILE/BUFFER?
          JMP      RECHD        ; NO, TRY AGAIN
RECLN     MVI      B,0           ;INIT. CHECKSUM
          CALL     HEXBIN       ;GET RECORD LENGTH
          ORA      A            ;IF RECLN=0, THEN DONE
          JZ      DWNLDNE       ; DONE
          STA      BUFCNT       ;SAVE THE RECLN
          MOV      C,A          ; NOT DONE - SAVE RECLN
          CALL     MDATAOUT      ;SEND IT TO MDS
          CALL     GETSADR       ;GET START ADDRESS

```

	LDA	FIRSTIME	; IF FIRST TIME THROUGH LOOP
	RRC		; THEN SAVE ADDR FOR LATER
	JC	RECLLEN1	; NOT FIRST TIME
	DCR	A	; SET THE FLAG
	STA	FIRSTIME	
	SHLD	START	; AND SAVE THE ADDRESS
RECLLEN1	SHLD	FINISH	; SAVE OTHER LOAD ADIRS
	CALL	ADDROUT	; SEND ADDRESS TO MDS
	XCHG		; GET BUFFER POINTER BACK
	CALL	HEXBIN	; IGNORE RECORD TYPE
HEXDATA	CALL	HEXBIN	; GET DATA BYTE
	CALL	MDATAOUT	; SEND DATA TO MDS
	DCR	C	; DECREMENT RECORD LENGTH
	JNZ	HEXDATA	; MORE TO GET
	CALL	CHECKIT	; SEE IF CKSUM IS OK
	INX	H	; GET NEXT RECORD
	JMP	RECHD	
DWNLDNE	LHLD	START	; GET STARTING LOAD ADDR
	PUSH	H	
	POP	B	; PREPARE IT FOR PRINTING
	LXI	H, DWNDONE1+20	
	CALL	CNVT16	
	LHLD	FINISH	; NOW READY THE FINISH ADDR
	LDA	BUFCNT	; GET RECLLEN
	ADD	L	
	MOV	L, A	
	MOV	A, H	
	ACI	0	
	MOV	H, A	
	PUSH	H	
	POP	B	
	LXI	H, DWNDONE1+43	
	CALL	CNVT16	
	LXI	D, DWNDONE	; PRINT COMPLETION MESSAGE
	CALL	PRINT	
	CALL	DELAY	
	CALL	HOSTDONE	; TELL MDS DONE
	JMP	MENU	
GETSADR	CALL	HEXBIN	; GET STARTING LOAD ADDRESS
	MOV	D, A	; FOR RECORD
	CALL	HEXBIN	
	MOV	E, A	
	XCHG		; HL = LOAD ADDRESS
			; IE = BUFFER POINTER
	RET		
CHECKIT	CALL	HEXBIN	; CHECK FOR CORRECT CHECKSUM
	XRA	A	
	ADD	B	; SHOULD BE ZERO
	RZ		; OK

```

                MVI      A,14                ;CHECKSUM ERROR
                JMP      ERROR

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

UPLD           MVI      A,128                ;INIT. BUFFER COUNT
                STA      BUFFCNT
                LXI      D,UPLDMSG          ;PRINT VERIFICATION MESSAGE
                CALL     PRINT
                CALL     GETFILEN           ;GET FILENAME & CHECK IT
                LXI      D,FCB
                CALL     DELETE             ;DELETE ANY EXISTING FILE
                CALL     CREATE             ;CREATE A NEW FILE
                CPI      255                ; CREATE OK?
                JNZ      UPLDØ1             ; YES
                MVI      A,16               ; NO, OUT OF DIRECTORY SPACE
                JMP      ERROR
UPLDØ1         CALL     BUFFERD            ;GET ADDRESS INPUTS
                MVI      A,1               ;ONE DELIMITER ALLOWED
                CALL     SCAN
                JNC      UPLD1              ;SCAN OK
                MVI      A,2               ;ERROR
                JMP      ERROR
UPLD1          CALL     GET4BIN             ;GET MDS START & FINISH
                SHLD     START              ; ADDRESSES FOR UPLOAD
                CALL     GET4BIN
                SHLD     FINISH
                XCHG                        ;DE = FINISH ADDRESS
                LHLD     START              ;CHECK FOR START > FINISH
                MOV      A,E
                SUB      L
                MOV      A,D
                SBB      H
                JNC      UPLD2              ; OK
                MVI      A,17              ;ERROR - START > FINISH
                JMP      ERROR
UPLI2          MVI      A,'U'              ;SEND UPLOAD CME TO MDS
                CALL     MDESCMD
                LHLD     START              ;SEND START & END ADDRESSES
                CALL     ADDRROUT
                LHLD     FINISH
                CALL     ADDRROUT
                LXI      H,DSKBUF
UPLI3          MVI      A,':'              ;STORE RECORD HEADER
                CALL     BUFFCK
                CALL     WRITLEN            ;STORE RECORD LENGTH
                CALL     WRITADDR          ;STORE STARTING LOAD ADDR
                CALL     WRITEATA          ;GET AND STORE DATA
                CALL     WRITCKS           ;STORE CHECKSUM & CR,LF

```

JMP	UPLD3	;DO ANOTHER HEX RECORD
WRITLNØ1 XRA	A	;WRITE LENGTH, ALTERNATE
JMP	WRITLEN1	; ENTRY FOR ZERO RECLEN
WRITLEN MVI	A,16	;ALL RECORDS HAVE RECLEN=16
		; EXCEPT THE LAST
WRITLEN1 MVI	B,Ø	;INIT. CHECKSUM
CALL	BINHEX	;CNVRT TC HEX ASCII & STORE
RET		
WRITADDR LDA	START+1	;STORE RECORD START ADDR
CALL	BINHEX	
LDA	START	
CALL	BINHEX	
PUSH	H	;SAVE BUFFER POINTER
LHLE	START	
LXI	D,16	;BUMP START ADDR FOR NEXT
DAD	D	; TIME
SHLE	START	
POP	H	;RESTORE BUFFER POINTER
XRA	A	;STORE RECORD TYPE
CALL	BINHEX	
RET		
WRITDATA MVI	C,16	;DATA COUNTER
WRITETA1 CALL	MDSIN	;GET DATA FROM MDS
LDA	MDSRDYF	;MORE DATA OR MDS DONE?
RRC		
JC	WRITINE	; MDS DONE
CALL	BINHEX	;MORE DATA
DCR	C	;16 BYTES YET?
RZ		; YES
JMP	WRITETA1	; NO, CONTINUE
WRITDNE XRA	A	
DCR	C	;FILL REMAINDER OF RECORD
JZ	WRITENØØ1	; WITH ZEROS
CALL	BINHEX	
JMP	WRITINE	
WRITENØØ1 CALL	WRITCKS	;STORE CHECKSUM
CALL	WRITEND	;STORE LAST RECORD
LDA	BUFFCNT	;IS BUFFER FULL?
MOV	B,A	
CPI	128	
JZ	WRITINE1	; YES
WRITENØ1 MVI	M,EOF	; NO, FILL REMAINDER WITH
INX	H	; EOF's
DCR	B	;DONE WITH FILL?
JNZ	WRITENØ1	; NO, CONTINUE
CALL	WRITEDSK	; YES, WRITE RECORD TO DISK
WRITINE1 CALL	CLOSEFILE	;CLOSE THE FILE
LXI	D,UPLDONE	;PRINT COMPLETION MESSAGE
CALL	PRINT	
CALL	DELAY	
JMP	MENU	

WRITCKS	MOV	A,B	;STORE CHECKSUM
	CMA		;GET NEGATIVE OF SUM
	INR	A	; ADD ONE
	CALL	BINHEX	
	MVI	A,CR	;STORE CR,LF SEQUENCE AT
	CALL	BUFFCK	; HEX RECORD END
	MVI	A,LF	
	CALL	BUFFCK	
	RET		
WRITEND	MVI	A,':'	;STORE LAST HEX RECORD
	CALL	BUFFCK	
	CALL	WRITLN01	;STORE 00 RECORD LENGTH
	XCHG		;DE = BUFFER POINTER
	LXI	H,0000H	;STORE 0000 LOAD ADDR &
	SHLD	START	; RECORD TYPE
	XCHG		;HL = BUFFER POINTER
	CALL	WRITADDR	
	CALL	WRITCKS	;STORE CHECKSUM
	RET		
BUFFCK	MOV	M,A	;STORE DATA
	INX	H	
	LDA	BUFFCNT	;IS BUFFER FULL?
	DCR	A	
	JZ	WRITEIT	; YES, SAVE IT ON DISK
	STA	BUFFCNT	; NO, SAVE COUNT
	RET		
WRITEIT	CALL	WRITEDSK	;WRITE 128 BYTE RECORD TO
			; DISK
	LXI	H,DSKBUFF	;REINIT. BUFFER AREA
	MVI	A,128	; AND BUFFER COUNT
	STA	BUFFCNT	
	RET		

* EXAM - EXAMINE/SET MDS MEMORY LOCATION(S) *

EXAM	LXI	D,EXAMSG	;PRINT VERIFICATION MESSAGE
	CALL	PRINT	
	CALL	BUFFRD	;GET ADDRESS INPUT
	XRA	A	;NO DELIMITERS ALLOWED
	CALL	SCAN	;DELIMITER CHECK
	JNC	EXAM01	; SCAN OK
	MVI	A,2	;INPUT ERROR (SYNTAX OR HEX)
	JMP	ERROR	
EXAM01	CALL	GET4BIN	;GET START ADDRESS
	SHLD	START	
	MVI	A,'X'	;SEND EXAM/SET CMD TO MDS
	CALL	MDSCMD	
	LHLD	START	
	CALL	ADDROUT	;SEND START ADDRESS TO MDS
EXAM1	CALL	MDSIN	;GET DATA IN MDS MEMORY
	STA	MISDATA	; SAVE IT

	PUSH	H	;SAVE ADDR. BEING EXAMINED
	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 MIS 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	
	JNC	EXAM02	; SCAN OK
EXAM020	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
	RAR		;PERIOD ONLY?
	JC	EXDONE	; YES - ALL DONE
	MVI	A,3	
	JMP	ERROR	; NO - PERIOD + DATA IS
			; ILLEGAL, START OVER
EXAM2	CALL	GET2BIN	
	MOV	A,L	;SEND NEW DATA
	JMP	SET1	
NOSET	LDA	MISDATA	;GET OLD DATA
SET1	CALL	MDATAOUT	
	POP	H	;BUMP ADDRESS FOR EXAM/SET
	INX	H	
	JMP	EXAM1	;GET MORE DATA FROM MIS
EXDONE	CALL	HOSTDONE	;SIGNAL MDS DONE
	JMP	MENU	;BACK TO MENU

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

FILL	LXI	D,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
SHLD    FINISH       ; SAVE IT TOO
CALL    GET2BIN      ;GET FILL DATA
MOV     A,L          ;A = DATA
STA     CONSIATA     ; SAVE IT
MVI     A,'F'        ;SEND FILL CME TO MDS
CALL    MDSCMD
LHLD    START        ;SEND START ADDR. TO MDS
CALL    ADDRROUT
LHLD    FINISH       ;SEND FINISH ADDR. TO MDS
CALL    ADDRROUT
LDA     CONSIATA     ;SEND FILL DATA TO MDS
CALL    MDATAOUT
MVI     A,1          ;MDS IN CONTROL
STA     SYSSTAT
FILL2   CALL    MDSIN ;MDS DONE FILLING?
XRA     A            ; YES - CLEAR FLAGS
STA     SYSSTAT
STA     MDSRDYF
JMP     MENU        ;RETURN TO MENU

```

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

```

ADDRROUT MOV    A,H      ;MSB FIRST
CALL    MDATAOUT
MOV     A,L            ; THEN LSB
CALL    MDATAOUT
RET                     ;BACK TO CALLER

```

* CSET - CONTINUOUS SET MDS MEMORY WITHOUT EXAMINE *

```

CSET     LXI     D,CSETMSG ;PRINT VERIFICATION MESSAGE
CALL    PRINT
MVI     A,0FFH          ;INIT. CONSOLE READ BUFFER
STA     CONBUFF         ; TO 255 CHARACTERS MAX
CALL    BUFRD           ;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
SHLD    START
MVI     A,'C'           ;SEND CSET CME TO MDS
CALL    MDSCMD
LHLD    START          ;SEND START ADDRESS TO MDS
CALL    ADDRROUT
JMP     CSET11
CSET1    CALL    CRLF
CSET11   CALL    BUFRD   ;GET REPLACEMENT DATA TILL
                        ; BUFFER FULL OR <CR>

```

	CALL	SCAN	;LOOK FOR ESCAPE
	CALL	CKPERIOD	;CHECK FOR PERIOD IN INPUT
	ORA	A	
	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 VALILATION DATA
CSET21	CALL	GET2BIN	;GET DATA
	MOV	A,L	
	CALL	MLATAOUT	;SEND IT TO MDS
	MOV	C,A	
	LXI	H,DATMSG+1	;SEND IT TO CONSOLE FOR
	CALL	CNVTS	; VERIFICATION
	XCHG		;LE = ALDR. OF DATMSG
	CALL	PRINT	
	XCHG		;LE = CURRENT CONBUFF PTR
	MOV	A,B	;AT END OF BUFFER?
	CPI	0FFH	
	JZ	CSET1	; YES, START OVER
	CPI	00	
	JZ	CSET1	; YES, START OVER
	JMP	CSET21	;NO, GET MORE DATA
CSET3	CALL	HOSTDONE	;NO DATA TO SEND, SIGNAL
			; MDS DONE
	JMP	MENU	;RETURN TO MENU

* EXEC - EXECUTE MDS MEMORY FROM A SPECIFIED ADDRESS *

EXEC	LXI	D,EXECMSG	;PRINT VERIFICATION MESSAGE
	CALL	PRINT	
	CALL	BUFFRD	;GET START ADDRESS
	XRA	A	;NO DELIMITERS ALLOWED
	CALL	SCAN	
	JNC	EXEC1	; SCAN OK
	MVI	A,2	;ERROR
	JMP	ERROR	;START OPTION OVER
EXEC1	CALL	GET4BIN	;GET START ADDRESS
	SHLD	START	; SAVE IT
	LXI	D,EXMSG	;SEE IF DATA FROM MDS TO
	CALL	PRINT	; CONSOLE OR NCT
EXEC11	CALL	CONSTAT	;WAIT FOR RESPONSE
	RRC		
	JNC	EXEC11	; LOOP
	CALL	CONSIN	;GET RESPONSE
	ORI	20H	;FORCE TO LOWER CASE
	CPI	'y'	;CONSOLE INPUT FROM MDS?
	JNZ	EXEC2	; NO, SEND CMI & RETURN TO
			; TO MENU

```

                LXI    D,EXMSG2      ;GIVE ESCAPE METHODS
                CALL   PRINT
                MVI    A,'E'         ; YES, SEND CML TO MIS &
                CALL   MDSCMD        ; LOOP WAITING FOR DATA
                LHLD   START         ; OR DONE FROM MIS OR ESC
                                   ; FROM CONSOLE
EXEC020 CALL   ADDR0UT
                CALL   MISIN         ;LOOP AT MISIN TILL ESC
                                   ; OR 'Q' OR DATA
                MOV    E,A           ;SAVE DATA FROM MDS
                LDA    MISRDYF       ;SEE IF MIS WANTS INPUT
                ORA    A
                JNZ    GETINP        ; YES
                CALL   CONSOUT       ; NO, SEND IT TO CONSOLE
                JMP    EXEC020       ; WAIT FOR MORE
GETINP  CALL   CONSTAT              ;GET INPUT FROM KEYBOARD
                RRC
                JNC    GETINP
                CALL   CONSIN
                CALL   MDATAOUT      ;SEND IT TO MIS
                XRA    A             ;RESET MISRDY FLAG
                STA    MDSRDYF
                JMP    EXEC020       ;LOOP AGAIN
EXEC2   MVI    A,'E'               ;SEND MDS EXEC CMD
                CALL   MDSCMD
                LHLD   START         ;SEND START ADDRESS TO MIS
                CALL   ADDR0UT
                MVI    A,1           ;SET MDS CONTROL FLAG
                STA    SYSSTAT
                JMP    MENU          ;BACK TO MENU

```

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

```

LOCATE  CALL   CLRBUF              ;CLEAR READ BUFFER
                LXI    D,LOCMSG     ;PRINT VERIFICATION MESSAGE
                CALL   PRINT
                MVI    A,0FFH       ;INIT. CONSOLE READ BUFFER
                STA    CONBUF       ; TO 255 CHARACTERS MAX
                CALL   BUFPRD       ;GET ADDRESS(ES)
                XRA    A             ;ANY DELIMITERS ?
                CALL   SCAN
                JNC    LOCATE1       ; NO, USE DEFAULT FINISH
                                   ; ADDRESS
                MVI    A,1           ;MORE THAN ONE DELIMITER?
                CALL   SCAN
                JNC    LOC01         ; NO, GET OPTIONAL FINISH
                                   ; ADDRESS
                MVI    A,2           ;MORE THAN 2 DELIMITERS
                JMP    ERROR         ; ERROR, START OPTION OVER
LOCATE1 CALL   GET4BIN              ;GET START ADDRESS
                SHLD   START

```

	JMP	LOC1	; NO COMMA, FINISH ADDRESS ; DEFAULTS TO 0FFFFH - ; GET DATA ;GET START ADDRESS
LOC21	CALL	GET4BIN	
	SHLD	START	
	CALL	GET4BIN	; COMMA, GET FINISH ADDRESS
	SHLD	FINISH	
	JMP	LOCDATA	
LOC1	LXI	H,0FFFFH	;SAVE DEFAULT FINISH ADDRESS
	SHLD	FINISH	
LOCDATA	MVI	A,'L'	;SEND LOCATE CME TO MDS
	CALL	MISCMD	
	LHLD	START	;SEND START ADDRESS TO MIS
	CALL	ADDROUT	
	LHLD	FINISH	;SEND FINISH ADDRESS TO MIS
	CALL	ADDROUT	
	MVI	A,16	:16 BYTES MAX
	PUSH	PSW	; SAVE BYTE COUNT
LOC1ATA1	CALL	BUFFRD	;GET SEARCH SEQUENCE
	CALL	SCAN	;LOOK FOR ESCAPE
	CALL	STAR	;PRINT A STAR
LOCDATA2	CALL	GET2BIN	;GET A BYTE
	MOV	A,L	
	CALL	MEATAOUT	;SEND IT TO MIS
	MOV	C,A	
	LXI	H,DATMSG+1	; & TO CONSOLE FOR
	CALL	CNVT8	; VERIFICATION
	XCHG		
	CALL	PRINT	
	XCHG		
	MOV	A,B	;AT END OF BUFFER?
	CPI	0FFH	
	JZ	LOC5	; YES, WAIT FOR SEARCH ; RESULTS
	CPI	00	
	JZ	LOC5	; YES, WAIT FOR SEARCH ; RESULTS
	POP	PSW	; NO, GET BYTE COUNT
	DCR	A	;16 BYTES YET?
	PUSH	PSW	;SAVE BYTE COUNT
	JZ	LOC5	; YES, WAIT FOR SEARCH ; RESULTS
	JMP	LOCATA2	; NOT AT END OR 16 BYTES
LOC5	CALL	HSTRDY	;TELL MDS TO SEARCH
	CALL	MISIN	;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,FOUNDMSG	;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,FOUNIMS1	;CONVERT ADDR. FOR PRINTING
XCHG		
CALL	PRINT	;PRINT ADDRESS
JMP	MENU	;BACK TO MENU

* DUMP - DUMP MDS MEMORY LOCATION(S) *

DUMP	CALL	CLRBUFF	;CLEAR READ BUFFER
	LXI	D,DUMPMMSG	;PRINT VERIFICATION MESSAGE
	CALL	PRINT	
	CALL	BUFFRD	;GET ADDRESS(ES)
	XRA	A	;ANY DELIMITERS?
	CALL	SCAN	
	JNC	DUMP01	; NO
	MVI	A,1	;MORE THEN ONE DELIMITER?
	CALL	SCAN	
	JNC	DUMP010	; NO, GET OPTIONAL FINISH
			; ADDRESS
	MVI	A,2	;MORE THAN ONE DELIMITER
	JMP	ERROR	; ERROR, START OPTION OVER
DUMP01	CALL	GET4BIN	;GET START ADDRESS
	SHLD	START	
	JMP	DUMP1	; NO COMMA
DUMP010	CALL	GET4BIN	;GET START ADDRESS
	SHLD	START	
	CALL	GET4BIN	;GET OPTIONAL FINISH ADDR
	SHLD	FINISH	
	JMP	DUMP2	
DUMP1	LHLD	START	;MAKE FINISH ADDRESS =
	LXI	B,0100H	; START + 256
	DAD	B	
	SHLD	FINISH	
DUMP2	MVI	A,'D'	;SEND DUMP CMD TO MDS
	CALL	MDSCMD	
	LHLD	START	;SEND START ADDRESS TO MDS
	CALL	ADDROUT	
	LHLD	FINISH	;SEND FINISH ADDRESS TO MDS
	CALL	ADDROUT	
DUMP3	LXI	D,DUMPMMSG3	;ASCII DATA STORAGE
	PUSH	D	
	CALL	MSG3INIT	;INIT. ASCII STORAGE
	CALL	MDSIN	;GET BYTE
	MOV	C,A	
	LDA	MDSRIYF	;MDS DONE TRANSMITING DATA?
	ORA	A	
	JNZ	DUMPDONE	; YES

	MOV	A,C	
	STA	MISDATA	; NO - SAVE DATA
	LHLD	START	;BC = START ADDRESS
	MOV	B,H	
	MOV	C,L	
	LXI	H,DUMPMMSG1	
	CALL	CNVT16	
	XCHG		
	CALL	PRINT	
	MVI	B,16	;SIXTEEN BYTES PER LINE
DUMPDATA	POP	D	;RECALL ASCII DATA STORAGE
			; LOCATION
	LDA	MISDATA	;GET DATA
	MOV	C,A	;IS DATA ASCII PRINTABLE?
	CPI	20H	
	JNC	IMPDTA1	; YES
	CALL	SPERIOD	; NO - STORE A PERIOD
	JMP	DMPDTA2	
IMPDTA1	CPI	80H	;GREATER THEN ASCII
	CNC	SPERIOD	; YES, STORE A PERIOD
DMPDTA2	STAX	D	; STORE DATA AS IS
	MOV	A,C	;RESTORE ORIGINAL DATA
	INX	D	;BUMP STORAGE ADDRESS
	PUSH	D	; AND SAVE IT
	LXI	H,DUMPMMSG2+1	;NOW CONVERT DATA TO HEX
			; AND PRINT IT
	PUSH	B	;SAVE COUNT
	CALL	CNVT8	
	XCHG		
	CALL	PRINT	
	POP	B	;GET COUNT BACK
	DCR	B	;16 BYTES YET?
	JZ	NXTLINE	; YES
	CALL	MDSIN	; NO - GET NEXT BYTE
	STA	MISDATA	;SAVE NEW DATA
	MOV	C,A	;MDS DONE TRANSMITING DATA?
	LDA	MDSRDYF	
	ORA	A	
	JNZ	NXTLINE	; YES
	JMP	DUMPDATA	; NO - GET NEXT LINE OF DATA
SPERIOD	MVI	A,'.'	;STORE A PERIOD IF NOT A
	RET		; PRINTABLE ASCII CHAR.
NXTLINE	LXI	D,DUMPMMSG3	;PRINT ASCII CHARACTERS
	CALL	PRINT	
	LXI	B,0010H	;GO TO NEXT LINE
	LHLD	START	;BUMP NEW LINE START ADDRESS
	DAL	B	; BY SIXTEEN BYTES
	SHLD	START	; SAVE IT
	POP	D	;GET GARBAGE OFF STACK
	LDA	MDSRDYF	;LONE?
	ORA	A	

	JNZ	DUMPDONE	; YES
	CALL	CRLF	; START NEW LINE
		DUMP3	; DUMP TILL DONE
DUMPDONE		A	; CLEAR MLS DONE XMITTING FLG
	S...	MDSRDYF	
	CALL	CRLF	; START NEW LINE
	CALL	BUFFRD	; ANOTHER DUMP?
	XRA	A	; NO DELIMITERS ALLOWED
	CALL	SCAN	
	JNC	DMPDNE1	; SCAN OK
	MVI	A,2	; ERROR
	JMP	ERROR	; START OPTION OVER
DMPDNE1	INX	D	; POINT TO END OF BUFFER
	DCR	B	; THERE YET?
	JNZ	DMPDNE1	; NO, LOOP
	LLAX	D	
	ORI	20H	; CONVERT TO LOWER CASE
	CPI	'd'	
	JZ	DUMPMORE	; YES - LUMP AGAIN FROM
			; PREVIOUS FINISH ALDR.
	JMP	MENU	; NO - RETURN TO MENU
DUMPMORE	LHLD	FINISH	; MAKE FINISH+1 = NEW START
	INX	H	; ADDRESS
	SHLD	START	
	JMP	DUMP1	; DUMP 256 MORE BYTES
MSG3INIT	MVI	B,17	; INIT. ASCII DATA STORAGE
	LXI	D,DUMPMMSG3	; AREA TO ALL \$'S
	MVI	A,'\$'	
MSG31	STAX	D	
	DCR	B	
	RZ		; INIT. DONE
	INX	D	
	JMP	MSG31	
CLRBUFF	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 MDS
	CALL	MESCMD	
	XRA	A	; RESET SYSTEM STATUS FLAG
	STA	SYSSTAT	
	LXI	D,ABORTEEM	; PRINT MLS ABORTED VERIFI-
	CALL	PRINT	; CATION
	CALL	DELAY	
	JMP	MENU	

*** UTILITY SUBROUTINES ***

* PRINT A STRING TO THE CONSOLE

* CALL WITH DE = STARTING ADDRESS OF STRING *

```
PRINT    PUSH    PSW                ;SAVE EVERYTHING
          PUSH    B
          PUSH    C
          PUSH    H
          MVI     C,PRSTRG          ;OUTPUT STRING TO CONSOLE
          CALL    BIOS
          POP     H
          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     D,SYSMSG+15
          JZ      SYS1            ;HOST IN CONTROL
          LXI     H,MDSMSG        ;MDS IN CONTROL
          JMP     SYS1+3          ;PUT 'MDS' IN MESSAGE
SYS1      LXI     H,HOSTMSG        ;PUT 'HOST' IN MESSAGE
          CALL    MOVESTR
          LDA     MENUSUPF        ;GET MENU SUPPRESSION FLAG
          ORA     A
          LXI     D,SYSMSG+33
          JZ      SYS3            ;NO SUPPRESSION
          LXI     H,YESMENMG      ;SUPPRESSION
          JMP     SYS3+3
SYS3      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     '$'            ;DE = DESTINATION ADDRESS
          RZ                  ;RETURN IF MOVE DONE
          STAX    D              ; NOT DONE
          INX     D
          INX     H
          JMP     MOVESTR        ;MOVE NEXT CHARACTER
```

* MENPMPT - PRINT MENU PROMPT *

```

MENPMPT LDA    MENUSUPP      ;SUPPRESS MENU?
        ORA    A
        JZ     MENPMT1      ; NO
        LXI    D,MENUPRO1    ; YES - PRINT SUPPRESSED
        CALL   PRINT        ; MENU PROMPT
        RET
MENPMT1 LXI    D,MENUPROM     ;PRINT UNSUPPRESSED MENU
        CALL   PRINT        ; PROMPT
        RET

```

** 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   CLRBUF      ;CLEAR CONSOLE INPUT BUFFER
        LXI    D,FILENAME   ;PROMPT FOR FILENAME
        CALL   PRINT
        CALL   BUFIRD      ;GET FILENAME
        CALL   FILENCK     ;DO CHECKS ON FILENAME
        ORA    A           ;SEE IF ANY ERRORS
        JZ     GETFN1      ; NO ERRORS
        CALL   ERROR       ; ERRORS
        JMP    GETFILEN    ;START OVER
GETFN1  CALL   MOVFN       ;MOVE FILENAME TO FCB
        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               ; DRIVE SELECTION
        JNC    FNCK2       ;SCAN OK
        MVI    A,8         ;TOO MANY COLONS
        RET
FNCK2  CALL   SCANUM       ;CHECK FOR TOO MANY OR TOO
        RRC               ; FEW CHARACTERS IN FILENAME
        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 FOR
RRC      ; 'HEX' FILETYPE
JNC      FNCK4        ;FILETYPE OK
MVI      A,10        ;ONLY 'HEX' FILETYPES ARE
RET      ; SUPPORTED
FNCK4    XRA      A    ;CHECK FOR ESCAPE AND
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
FNCK6    XRA      A    ;NO ERRORS DETECTED
RET      ; FILENAME OK

```

```

* SCANQ - SCAN FILENAME FOR QUESTION MARKS INDICATING AN
*          AMBIGUOUS FILENAME
* RETURN  A = 00 IF NONE FOUND
*          = 0FFH IF FOUND *

```

```

SCANQ    PUSH      B
        PUSH      C
        PUSH      H
        XCHG      ;HL = BUFFER + 1
        MOV       C,M ;GET BUFFER 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,0FFH ;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
*           = 0FFH IF AN ILLEGAL ':' IS FOUND *

```

```

SCANCOL  PUSH      B

```

PUSH	D	
PUSH	H	
CALL	CURDSK	;GET CURRENT LISK
ORI	40H	;CONVERT IT TO A CHARACTER
INR	A	
STA	CURRENT	;SAVE IT
XCHG		;GET BUFFER COUNT
MOV	C,M	
INX	H	
INX	H	;THE ONLY ':' WOULD BE HERE
DCR	C	
MOV	A,M	
CPI	:	
JNZ	SCANCOL1	;NONE HERE
DCX	H	;FOUND IT, CHECK FOR
INR	C	; CORRECT DRIVE
MOV	A,M	
ANI	0IFH	;FORCE TO UPPER CASE
MOV	B,A	
LDA	CURRENT	
CMP	B	;SAME?
JZ	SCNCOL11	; YES, OK
LXI	D,DRIVERR	; NO, PRINT WARNING &
CALL	PRINT	; IGNORE IT
CALL	DELAY	
INX	H	
DCR	C	
JMP	SCANCOL2	;CONTINUE SCAN
SCANCOL1 DCX	H	;CHECK IF 1st CHAR IS ':'
INR	C	
SCNCOL11 MOV	A,M	
CPI	:	
JZ	SCANCOL3	; YES, ERROR
DCR	C	; NO
JZ	SCNCOLDN	;SCAN DONE
INX	H	;SCAN NOT DONE
DCR	C	
SCANCOL2 INX	H	
MOV	A,M	;SEE IF ANY MORE ':'
CPI	:	
JZ	SCANCOL3	; YES, ERROR
DCR	C	; NO
JNZ	SCANCOL2	;CONTINUE SCAN
SCNCOLDN XRA	A	;DONE, NO ERRORS DETECTED
JMP	SCANCOL3+2	
SCANCOL3 MVI	A,0FFH	;TOO MANY ':'
POP	H	
POP	C	
POP	B	
RET		

```

* SCANHEX - SCAN FILETYPE FOR 'HEX'
* RETURN  A = 00 IF FOUND
*          = 0FFH IF NOT FOUND *

```

```

SCANHEX  PUSH    B
        PUSH    D
        PUSH    H
        XCHG                    ;GET BUFFER COUNT
        MOV     C,M
SCANHX1  INX     H                ;GO TO PERIOD
        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     0DFH             ;FORCE TO UPPER CASE
        CPI     'H'
        JNZ     SCNHXER          ;ERROR
        INX     H
        MOV     A,M
        ANI     0DFH
        CPI     'E'
        JNZ     SCNHXER
        INX     H
        MOV     A,M
        ANI     0DFH
        CPI     'X'
        JNZ     SCNHXER
        XRA     A                ;NO ERROR
        JMP     SCNHXER+2
SCNHXER  MVI     A,0FFH          ;ERROR
        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
*          = 0FFH IF ERROR *

```

```

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

```

```

SCANUM1 INX      H
        MOV      A,M
        CPI      ':'
        JNZ      SCANUM2      ; START COUNT AT ':'?
        DCR      B            ; YES
        ICR      C            ; NC, START AT BEGINNING
        JZ       SCANUM4      ; DONE YET?
        JMP      SCANUM1      ; YES
        JMP      SCANUM1      ; NO
SCANUM2 CPI      PERIOD      ; GO TO PERIOD OR BUFFER END
        JZ       SCANUM4      ; PERIOD, DONE
        INR      B            ; KEEP COUNTING
        ICR      C
        JZ       SCANUM4      ; DONE
        JMP      SCANUM1      ; LOOP
SCANUM4 XRA      A            ; < 1 CHARACTER?
        CMP      B
        JZ       SCANUM5      ; YES, ERROR
        MVI      A,8          ; > 8 CHARACTERS?
        CMP      B
        JC       SCANUM5      ; YES, ERROR
        XRA      A            ; NO ERRORS
        JMP      SCANUM5+2
SCANUM5 MVI      A,0FFH      ; ERROR
        POP      H
        POP      D
        POP      B
        RET

```

```

* SCANINV - SCAN FILENAME FOR NON-PRINTABLE CHARACTERS
* RETURN  A = 00 IF NONE FOUND
*          = 0FFH IF ANY FOUND *

```

```

SCANINV PUSH     B
        PUSH     D
        PUSH     H
        XCHG                    ; GET BUFFER COUNT
        MOV      C,M
SCANIN1 INX      H
        MOV      A,M
        CPI      20H            ; < SPACE?
        JC       SCANIN2      ; YES, ERROR
        DCR      C            ; DONE WITH SCAN?
        JNZ      SCANIN1      ; NO
        XRA      A            ; YES, NO ERRORS
        JMP      SCANIN2+2
SCANIN2 MVI      A,0FFH      ; ERROR
        POP      H
        POP      D
        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                    ;IE = CONEUFF POINTER
       INX     D
       INX     D              ;SEE IF IT'S A COLON
       DCR     C
       LDAX    D
       CPI     ':'
       JZ      MOVIT01       ; YES
       DCX     D              ; NO
       INR     C
       JMP     MOVIT         ;START AT BUFFER START
MOVIT01 INX     D              ;START FROM COLON
       DCR     C
MOVIT  LXI     H,FCB+1
MOVIT1 LDAX    D              ;MOVE THE FILENAME
       CPI     PERIOD        ; UNTIL PERIOD OR END
                                ; OF BUFFER
       RZ                      ;DONE
       MOV     M,A            ;STORE CHAR. IN FCB
       INX     H
       INX     D
       DCR     C              ;AT END OF BUFFER?
       RZ                      ; 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,FCB
       LXI     D,FCBMSG
       MVI     C,16          ;SET UP FIRST 16 BYTES
PURG01  LDAX    D
       MOV     M,A
       DCR     C              ;16 BYTES DONE YET?
       JZ      PURG1         ; YES
       INX     H
       INX     D
       JMP     PURG01        ; NO, LOOP
PURG1   LXI     D,FCB+32     ;INITIALIZE CURRENT RECORD
       XRA     A              ; BYTE IN FCB
       STAX    D
       RET

```

* UCASE - CONVERT ALL FILENAME ALPHABETICS TO UPPER CASE *

```

UCASE  MVI     C,8           ;8 CHARACTERS MAX
       LXI     H,FCB+1

```

```

UCASE01  MOV     A,M
          CPI     7BH                ;IS IT > LOWERCASE z?
          JNC     UCASE1             ; YES, OK
          CPI     'a'                ; NO, IS IT < LOWERCASE a?
          JC      UCASE1             ; YES, OK
          ANI     0DFH               ;MUST BE LOWER CASE
          MOV     M,A                ; CONVERT IT TO UPPER CASE
UCASE1   INX     H
          DCR     C                  ;DONE?
          RZ                      ; YES
          JMP     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     H
          MOV     A,M                ;GET FIRST DIGIT
          CALL    EOFCK              ;END OF BUFFER/FILE?
          CALL    ASCHX              ;CONVERT TO PURE HEX
          RLC                      ;MAKE IT 4 MSB'S
          RLC
          RLC
          RLC
          MOV     E,A                ;SAVE IT
          INX     H                  ;GET SECOND DIGIT
          MOV     A,M
          CALL    EOFCK
          CALL    ASCHX              ;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

```

```

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

```

```

ASCHX    SUI     '0'                ;SUBTRACT OFF ASCII BIAS
          CPI     10
          RC                      ;NUMBER IS 0-9
          SUI     7                  ;NUMBER IS A-F
          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 *

```

AD-A104 684

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/G 9/2

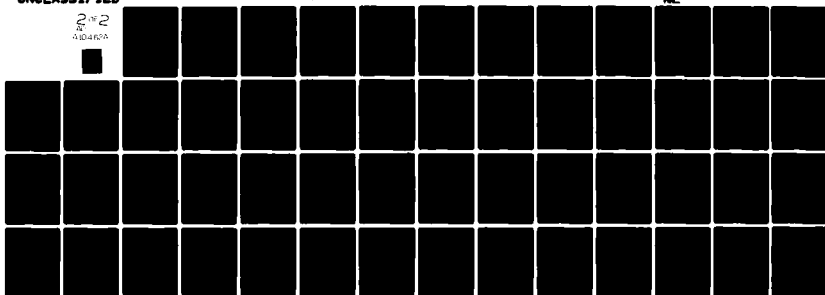
A MICROPROCESSOR DEVELOPMENT SYSTEM FOR THE ALTOS SERIES MICROC-ETC(U)

JUN 81 S M HUGHES

UNCLASSIFIED

NL

2 of 2
0.004604



END

DATE

FORMED



DTIC

```

EOFCK  CPI      EOF
        RNZ
        LDA      CONTF LG      ;NOT END OF FILE/BUFFER
        RRC      ;SEE IF END OF FILE
        JNC      DWNLDNE      ; YES
        LXI      H,DSKBUFF    ; NO, READ MORE
        CALL     READSK
        LXI      H,DSKBUFF
        MOV      A,M
        RET

```

```

* BINHEX - CONVERT AN EIGHT BIT BINARY NUMBER TO TWO HEX
*         ASCII CHARACTERS
*         - STORE THE CHARACTERS IN MEMORY POINTED TO BY HL
*         - ADD BINARY NUMBER TO RUNNING CHECKSUM IN D
* CALL WITH BINARY NUMBER IN A AND HL AS ABOVE *

```

```

BINHEX  PUSH     PSW           ;SAVE DATA
        ADD      B            ;ADD 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
        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          ;ADD 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
READSK1	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	DCR	B	;4K WORTH YET?
	JNZ	READSK1	; NO, READ MORE
	MVI	M,EOF	; YES, STORE END OF BUFFER
			; INDICATOR
	MVI	A,0FFH	;SET CONTINUATION FLAG
	STA	CONTF LG	
	POP	B	;RESTORE B
	RET		
READNE	XRA	A	;RESET CONTINUATION FLAG
	STA	CONTF LG	
	PUSH	B	
	LXI	B,-128	;POINT TO END OF LAST RECORD
	DAL	B	
	POP	B	
	MVI	M,EOF	;ENSURE EOF MARKER IN BUFFER
	POP	B	;RESTORE ORIGINAL B
	RET		

* WRITEDSK - WRITE A SINGLE 128 BYTE RECORD TO DISK *

WRITEDSK	LXI	H,DSKBUF	;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	
	CALL	CLOSEFILE	;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	D
	PUSH	H
	LXI	D,FCB
	MVI	C,READF
	CALL	BIOS
	POP	H
	POP	D

```

        POP    B
        RET

```

* WRITEREC - WRITE A SINGLE RECORD TO DISK *

```

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

```

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

```

DMASET    PUSH    PSW
          PUSH    B
          PUSH    D
          PUSH    H
          KCHG
          MVI     C,SETDMA
          CALL    BDOS
          POP     H
          LXI     B,128
          DAD     B
          POP     D
          POP     B
          POP     PSW
          RET

```

;DE = DMA ADDRESS

;READY DMA ADDRESS FOR NEXT
; TIME

* OPENFILE - OPEN A FILE CURRENTLY ON DISK *

```

OPENFILE  PUSH    B
          PUSH    D
          PUSH    H
          LXI     D,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    D
        PUSH    H
        LXI     D,FCB
        MVI     C,MAKEF
        CALL    BDOS
        POP     H
        POP     D
        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,CURRNTD
        CALL    BDOS
        POP     H
        POP     D
        POP     B
        RET

```

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

```

ERROR    MOV    C,A           ;GET ERROR NUMBER
          MVI    B,0           ;COMPUTE ERROR VECTOR
          LXI    H,ERRJMP-3
          LAD    B
          DAD    B
          LAL    B
          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       ;HEX 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       ;DISK READ ERROR
          JMP    ERROR16       ;OUT OF DIRECTORY SPACE
          JMP    ERROR17       ;START > FINISH ADDRESS
          JMP    ERROR18       ;OUT OF DIR/DISK SPACE
                                   ; PARTIAL FILE NOT SAVED

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

ERROR2   LXI    D,MFILELERR    ;PRINT ERROR MESSAGE
          JMP    ERROUT1

ERROR3   LXI    D,PERONLYM
          JMP    ERROUT1

ERROR4   LXI    D,INVHEXER
          JMP    ERROUT1

ERROR5   LXI    D,SECELELERR
          JMP    ERROUT1

ERROR6   LXI    D,SEQDELER
          JMP    ERROUT1

ERROR7   LXI    D,AMBIGERR

```

	JMP	ERRROUT2	
ERROR8	LXI	D,COLONERR	
	JMP	ERRROUT2	
ERROR9	LXI	D,FNCHARERR	
	JMP	ERRROUT2	
ERROR10	LXI	D,HEXFTErr	
	JMP	ERRROUT2	
ERROR11	LXI	D,SPFNERR	
	JMP	ERRROUT2	
ERROR12	LXI	D,NPRTErr	
	JMP	ERRROUT2	
ERROR13	LXI	D,FNFNDERR	
	JMP	ERRROUT1	
ERROR14	LXI	D,CKSUMERR	
	JMP	ERRROUT3	
ERROR15	LXI	D,DSKRDERR	
	JMP	ERRROUT3	
ERROR16	LXI	D,DIRSPERR	
	JMP	ERRROUT	
ERROR17	LXI	D,SGFAERR	
	JMP	ERRROUT1	
ERROR18	LXI	D,DDSPCERR	
	JMP	ERRROUT3	
ERRROUT	CALL	PRINT	;PRINT ERROR
	CALL	DELAY	;LET USER READ ERROR
	JMP	MENU	;START OVER
ERRROUT1	LXI	SP,STACK	;RE-INIT. STACK
	CALL	PRINT	;PRINT ERROR
	CALL	DELAY	
	LDA	MENUFLG	;RECALL MENU CHOICE
	JMP	MENU1	;RESTART CURRENT OPTION
ERRROUT2	CALL	PRINT	;PRINT ERROR
	CALL	DELAY	
	RET		;BACK TO CALLER
ERRROUT3	CALL	PRINT	;PRINT ERROR
	CALL	DELAY	

```

CALL    HOSTDONE    ;TELL MES DONE
JMP     MENU

```

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

```

DELAY   PUSH    PSW
        PUSH    B
        PUSH    C
        PUSH    H
        MVI     B,15          ;OUTER LOOP INITIALIZATION
        LXI     D,-1         ;DECREMENT BY SUBTRACTION
DELAYIN LXI     H,39E0H       ;INNER LOOP INITIALIZATION
DELAYOUT DAD     D           ;HL = HL - 1
        JC      DELAYOUT
        DCR     B
        JNZ     DELAYIN
        POP     H
        POP     C
        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
 * IE = 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  LDAX    D             ;GET DATA FOR CONVERSION
        CPI     'A'          ;IS IT 0-9?
        JC      ENTER15      ; YES
        ANI     01FH         ; NO - FORCE TO UPPER CASE
ENTER15 DAD     H             ;SHIFT PREVIOUS DATA LEFT
        DAD     H             ; 4 BITS
        DAD     H
        DAD     H
        JC      ENTER3       ;IF OVERFLOW, PRINT ERROR

```

```

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
                     ; ERROR
                     ;START OVER
ENTER4   JMP      ERROR ;RESTORE REGISTERS
POP      D
POP      B
POP      PSW
RET

```

* CONFIN - CONSOLE INPUT ROUTINE
 * DOESN'T RETURN UNTIL INPUT IS RECEIVED *

```

CONFIN   PUSH      B           ;SAVE REGISTERS
        PUSH      D
        PUSH      H
        MVI      C,CONIN      ;GET CHARACTER
        CALL     BIOS
        POP       H           ;RESTORE REGISTERS
        POP       D
        POP       B
        RET                ;RETURN TO CALLER WITH
                          ; CHARACTER IN A

```

* CONSOUT - CONSOLE OUTPUT ROUTINE
 * ENTER WITH CHARACTER IN E *

```

CONSOUT  PUSH      PSW        ;SAVE REGISTERS
        PUSH      B
        PUSH      D
        PUSH      H
        MVI      C,CONSOUT    ;OUTPUT CHARACTER
        CALL     BDOS
        POP       H           ;RESTORE ALL REGISTERS
        POP       D
        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     D
          POP     B
          RET

```

```

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

```

```

BUFFRD  PUSH    PSW           ;SAVE A, HL
          PUSH    H
BUFF1   LXI     D,PROMPT      ;SEND PROMPT TO CONSOLE
          CALL    PRINT
          LXI     D,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     D
          LDAX   D           ;GET COUNT
          ORA     A           ;IS COUNT = 0?
          JNZ     READONE      ;NO, RETURN TO CALLER
          JMP     BUFF1        ;YES, TRY AGAIN
READONE MOV     B,A           ;RETURN WITH B = COUNT
          POP     H           ;RESTORE A, HL
          POP     PSW
          RET

```

```

* BUFFRD1 - READ CONSOLE INPUT INTO BUFFER POINTED TO BY DE
* RETURN WITH DE = BUFFER START ADDRESS + 1
*             B = COUNT OF CHARACTERS INPUT
*             A = 00 IF COUNT = 0
*             = 0FFH IF COUNT > 0
*             HL UNCHANGED *

```

```

BUFFRD1 PUSH    H           ;SAVE HL
          LXI     D,CONBUFF    ;POINT TO CONSOLE BUFFER
          PUSH    D           ;SAVE IT
          MVI     C,READCON    ;READ CONSOLE INPUT
          CALL    BIOS

```

```

CALL    CRLF
POP     C                    ;POINT TO CHAR. COUNT
INX     D
LDAX    D                    ;GET COUNT
ORA     A                    ;IS COUNT = 0?
JZ      READONE1             ; YES, RETURN TO CALLER
MOV     B,A                  ;SAVE CHAR COUNT
MVI     A,0FFH               ;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    PUSE    B            ;SAVE REGISTERS
        PUSH    C
        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
                             ;SCAN FOR SEQUENTIAL DELS.
SCAN1    CALL    SCANDEL
        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      SCANESC      ; YES, ESCAPE FROM OPTION
        CPI     '?'          ;IS IT A QUEST FOR HELP?
        JZ      QUESTION    ; YES, PRINT DATA FORMATS
SCAN2    DCR     C            ;NONE OF THESE, CHECK NEXT
                             ; CHARACTER
        JZ      SCANLONE     ;NO MORE CHARACTERS TO CHECK
        JMP     SCAN1
CNTDEL   DCR     B            ;DECREMENT DELIMITER COUNT
        JMP     SCAN2        ;LOOK FOR ANOTHER DELIMITER
SCANLONE XRA     A            ;SEE IF B = 0
        CMP     B
SCAND1   POP     H            ;RESTORE REGISTERS

```

```

      POP      D
      POP      B
SCANESC LDA      MENUFLG      ;IF HOST COMMAND THEN
      CPI      NHSTCML      ; NO ESCAPE TO MDS
      JC       MENU
      LDA      SYSSTAT      :SEE IF HOST IN CONTROL
      ORA      A
      JZ       SCNESC1      ;HOST IN CONTROL
      CALL     CNTRLCK      ;MDS ID IN CONTROL
      RRC
      JNC      MENU        ; NO ABORT
SCNESC1 MVI      A,'Q'      ; ABORT
      CALL     MDSCMD
      XRA      A           ;CLEAR SYSSTAT FLAG, HOST
      STA      SYSSTAT      ; NOW IN CONTROL
      JMP      MENU        ;RETURN TO MENU

QUESTION LXI      D,FORMTMSG      ;PRINT DATA FORMATS AND
      CALL     PRINT      ; RETURN TO CURRENT OPTION
QUEST1  CALL     CONSTAT      ;WAIT FOR RESPONSE TO
      RRC      ; CONTINUE
      JNC      QUEST1
      CALL     CONSIN
      LXI      D,FMTMSG1      ;CONTINUE FORMAT MESSAGE
      CALL     PRINT
QUEST2  CALL     CONSTAT
      RRC
      JNC      QUEST2
      CALL     CONSIN
      LDA      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
      CALL     CONSIN      ;GET RESPONSE
      ORI      20H      ;FORCE IT TO LOWER CASE
      CPI      'Y'      ;ABORT MDS CONTROL?
      JZ       CNTRL2      ; YES
      XRA      A           ; NO, CLEAR A
      RET
CNTRL2  MVI      A,0FFH      ; SET A
      RET

```

```

* SCNENDEL - CHECK FOR DELIMITERS AT FIRST & LAST CHARACTER
*           POSITIONS IN CONSOLE INPUT BUFFER
* CALL WITH BUFFER COUNT IN C *

```

```

SCNENDEL 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     ; NC, 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 - SCAN CONSOLE BUFFER FOR 2 OR MORE SEQUENTIAL
* DELIMITERS *

```

SCANDEL PUSH    B           ;SAVE BUFFER COUNT
                XRA          A           ;INIT. FIRST DELIMITER FLAG
                STA          FRSTDEL
SDEL1 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           SDELDNE
                XRA          A           ;RESET FLAG
                STA          FRSTDEL
                JMP          SDEL1       ;LOOP
DELCK LDA        FRSTDEL     ;FIRST DELIMITER?
                ORA          A
                JNZ          DELCK1     ; NO, A=1 - ERROR
                INR          A           ; YES, SET FRSTDEL FLAG
                STA          FRSTDEL
                DCR          C           ;SEE IF DONE
                JZ           SDELDNE
                JMP          SDEL1       ; NO, LOOP
DELCK1 XRA       A           ;PRINT ERROR
                STA          FRSTDEL
                MVI          A,6
                JMP          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 + 1
* RETURN WITH A = 00 IF NO PERIOD FOUND
*              = 0FFH IF A PERIOD ONLY
*              = 0F0H IF A PERIOD + DATA
*              OTHER REGISTERS UNCHANGED *

```

```

CKPERIOD PUSH  B           ;SAVE REGISTERS
             PUSH  D
             PUSH  H
             XCHG           ;HL = CONBUFF + 1
             MOV   C,M       ;C = CHARACTER COUNT
             MOV   D,M       ;D = 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,D         ;RECALL ORIG. CHAR. COUNT
             CPI   1        ;ONLY A PERIOD?
             JZ    NOERR    ; YES, NO ERROR
             MVI   A,0F0H   ; PERIOD + DATA IS ILLEGAL
             JMP   CKDONE+1
NOERR  MVI     A,0FFH      ;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, GO 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   XCHG      ;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
        DCR      B            ;MORE CHARACTERS IN BUFFER?
        JZ       BACKUP0      ; NO
        JMP      GET4LOOP     ;NONE OF THESE, TRY AGAIN
BACKUP0 INX      H           ;POINT TO BUFFER END + 1
BACKUP  PUSH     H           ;SAVE DELIMITER ADDRESS
        DCX      H            ;BACK UP 1
        CALL     BUFFTST      ;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
        DCR      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      ;DE = CONVERSION START ADDR
        CALL     ENTER       ;DO CONVERSION
        POP      D            ;IE = DELIMITER ADDRESS
        DCR      B            ;DECREMENT CHAR. COUNT
        RET
BUFFTST PUSH     H
        PUSH     D
        LXI      D,CONBUFF+1
        MOV      A,L         ;AT BUFFER+1 YET?
        CMP      E           ;IF Z = 1 THEN AT BUFFER+1
        POP      L
        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

```

;TWO BACK-UP'S ONLY
; MODIFY GET4BIN CODE
;RESTORE GET4BIN CODE

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

```

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

```

;SAVE REGISTERS
;SAVE CHARACTER
;RESET SIO INT BIT
;GET SIO STATUS
;CHECK FOR BOTH DTR & TXE
; MUST HAVE BOTH
;LOOP TILL READY
;SEND CHARACTER
;IF XON, DON'T WAIT FOR
; CONFIRMATION
;NOW WAIT FOR CONFIRMATION
; FROM MDS
;GET IT TO RESET SIO FLAGS
;RESTORE REGISTERS

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

```

MDSCMD  PUSH    PSW
        MVI    A,055H
        CALL   MDSOUT
        POP     PSW
        CALL   MDSOUT
        RET

```

;SAVE COMMAND
;NEXT CHAR. WILL BE CMD
;SEND COMMAND

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

```

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

```

;SAVE DATA
;NEXT CHAR. WILL BE DATA
;SEND DATA

```

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

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

HOSTRDY MVI     A,00H              ; NEXT CHAR. IS RDY FLAG
        CALL    MDSOUT
        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 CMND
        CALL    MDSCMD
        RET

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

MDSIN   PUSH    B                  ; SAVE REGISTERS
        PUSH    D
        PUSH    H
        CALL    MDSINRDY           ; ANY INPUT WAITING FROM MDS?
        IN      MDATAPT            ; YES, GET DATA TYPE
        CPI     0FFH               ; IS IT DATA?
        JZ      MDSIN2             ; YES, GET IT
        CPI     055H               ; QUIT CMD?
        JZ      MDSQUIT            ; YES
        JMP     MDSINENE           ; NO, MDS MUST HAVE
                                      ; SIGNALLED IT'S READY
                                      ; FOR INPUT
                                      ; CONFIRM RECEIPT

MDSQUIT MVI     A,XON
        CALL    MDSOUT
        CALL    MDSINRDY
        IN      MDATAPT
        XRA     A                  ; RESET FLAGS
        STA     SYSSTAT
        STA     MDSRLYF
        MVI     A,XON              ; CONFIRM RECEIPT OF 'Q'
        CALL    MDSOUT
        JMP     MENU               ; NOW BACK TO MENU

MDSIN2  MVI     A,XON              ; SEND CONFIRMATION
        CALL    MDSOUT
        CALL    MDSINRDY           ; WAIT FOR DATA
        IN      MDATAPT            ; THEN GET IT
        PUSH    PSW                ; SAVE IT

```

```

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

* MDSINDNE - SET MDS READY FOR INPUT FLAG *

MDSINDNE MVI      A,XON        ;CONFIRM IT
CALL     MDSOUT
CALL     MDSINRDY
IN       MDATAPT
MVI      A,0FFH           ;SET MDS READY FLAG
STA      MDSRDYF
MVI      A,XON            ;CONFIRM RECEIPT OF DATA
CALL     MDSOUT
POP      H                ;RESTORE REGISTERS
POP      D
POP      B
RET                        ;BACK TO MDSIN CALLER

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

ESCK     CALL     CONSTAT      ;CHECK FOR INPUT
RRC
RNC
CALL     CONSIN              ;IS IT ESCAPE?
CPI      ESC                ;IS IT ESCAPE?
JZ       ESCK01              ; NO
MVI      E,BKSPCE           ;DON'T PRINT CHARACTER
CALL     CONSOUT
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              ; TO MDS

```

```

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

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

MSTAT    XRA      A                ;CHECK SIO STATUS
          OUT     MSTATPT
          IN      MSTATPT
          ANI     1                ;CHARACTER WAITING?
          RZ                      ; NO, RETURN WITH A = 0
          MVI     A,0FFH          ; YES, RETURN WITH A = 0FFH
          RET

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

CNVT16    PUSH    H                ;SAVE REGISTERS
          PUSH    D
          PUSH    B
          INX     H
          INX     H
          INX     H
          MVI     D,4              ;CHARACTER COUNTER
CNVT161   MOV     A,C              ;NEXT 4 BITS
          ANI     0FH
          CPI     0AH              ;IS IT A-F?
          JC      CNVT1615         ; NO
          ADI     7                ; YES
CNVT1615  ADI     '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
          RAR
          MOV     B,A
          MOV     A,C
          RAR
          MOV     C,A
          ICR     E                ;DECREMENT SHIFT COUNTER
          JNZ     CNVT162         ;STILL SHIFTING
          ICR     L                ;DECREMENT CHARACTER COUNTER
          JNZ     CNVT161         ;STILL CONVERTING

```

```

      POP      B           ;RESTORE REGISTERS
      POP      C
      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     D,STARMSG    ;PRINT IT
      CALL   PRINT
      POP     D
      RET                     ;BACK TO CALLER

```

```

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

```

```

SIGNON  DB      CR,LF,'ALTOS MDS 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'
        DB      'CASE.',CR,LF
        DB      'C. ADDRESS AND DATA INPUTS ARE EXPECTED'
        DB      'TO BE IN HEX NOTATION.',CR,LF
        DB      'D. TERMINATE INPUTS WITH A CARRIAGE '
        DB      'RETURN OR LINE FEED.',CR,LF
        DB      'E. NORMAL LINE EDITING ON INPUT IS AS '
        DB      'IN CP/M AND MP/M.',CR,LF
        DB      'F. FOR ADDRESS INPUTS, THE PROGRAM '
        DB      'WILL ALWAYS TAKE THE LAST FOUR OR LESS '
        DB      CR,LF,'HEX CHARACTERS ENTERED; FOR '
        DB      'DATA INPUTS, THE LAST TWO OR LESS.',CR,LF
        DB      'G. SOURCES OF COMMON ERROR ARE INVALID'
        DB      'HEX DIGITS, TOO MANY OR TOO FEW',CR,LF
        DB      'DELIMITERS, AND ILLEGAL SYNTAX.',CR,LF

```

```

EB      ' H. IN GENERAL, THE SAME DATA I/O FORMAT'
EB      ' AS USED IN DIGITAL RESEARCH'S',CR,LF
EB      ' DET IS USED HERE. FOR EXCEPTIONS,'
EB      ' CONSULT THE USER'S MANUAL.',CR,LF
DB      ' I. A QUESTION MARK ENTERED AFTER THE '
EB      ' PROMPT WILL CAUSE THE INPUT FORMATS TO'
EB      CR,LF
DB      ' BE DISPLAYED.',CR,LF
EB      ' J. IF THE ESCAPE KEY IS ENTERED DURING '
EB      ' INPUT THEN THE USER IS RETURNED',CR,LF
DB      ' TO THE MENU.',CR,LF
EB      ' K. FOR FURTHER DETAILS, CONSULT THE '
EB      ' USER'S MANUAL',CR,LF,LF
DB      ' PRESS ANY KEY TO CONTINUE >$'

MENUMSG EB      CR,LF,'
DB      ' MENU',CR,LF
EB      ' HOST COMMANDS
EB      ' MDS COMMANDS',CR,LF,LF
DB      ' A. SUPPRESS PRINTING MENU
EB      ' G. DOWNLOAD HEX FILE - DISK TO MDS
EB      ' MEMORY',CR,LF
DB      ' B. DO NOT SUPPRESS PRINTING MENU
EB      ' H. UPLOAD MDS MEMORY TO HEX DISK FILE'
EB      CR,LF
DB      ' C. BASIC INSTRUCTIONS
EB      ' I. EXAMINE/SET MDS MEMORY LOCATION(S)'
EB      CR,LF
DB      ' D. HEXADECIMAL ADD & SUBTRACT
EB      ' J. CONTINUOUS SET OF MDS MEMORY',CR,LF
EB      ' E. RETURN SYSTEM CONTROL TO HOST
DB      ' K. FILL MDS MEMORY WITH SPECIFIED BYTE'
EB      CR,LF
EB      ' F. RETURN TO CP/M
DB      ' L. LOCATE BYTE SEQUENCE IN MDS MEMORY'
EB      CR,LF
EB      '
DB      ' M. DUMP MDS MEMORY LOCATION(S) TO CONSOLE'
EB      CR,LF
EB      '
DB      ' N. EXECUTE MDS MEMORY FROM SPECIFIED',CR,LF
EB      '
EB      ' LOCATION',CR,LF,'$'
SYSMSG  DB      ' SYSTEM STATUS: $$$ IN CONTROL;'
EB      ' $$ MENU SUPPRESSION',CR,LF,'$'
MISMSG  EB      ' MDS $'
HOSTMSG DB      ' HOST$'
NOMENMSG EB      ' NO$'
YESMENMG EB      '$'
MENERRMG DB      CR,LF,'INVALID MENU SELECTION',CR,LF,'$'
MFELEERR EB      CR,LF,'TOO MANY OR TOO FEW DELIMITERS IN'

```

	DB	' INPUT',CR,LF,'\$'
PERONLYM	DB	CR,LF,'PERIOD ONLY PLEASE !',CR,LF,'\$'
INVHEXER	DB	CR,LF,'INVALID HEX DIGIT',CR,LF,'\$'
SEDELEERR	DB	CR,LF,'CAN'T HAVE A DELIMITER AT START OR
	LB	' END OF INPUT',CR,LF,'\$'
SEQDELER	DB	CR,LF,'TWO OR MORE DELIMITERS SEQUENTIALLY'
	DB	CR,LF,'\$'
AMBIGERR	DB	CR,LF,'AMBIGUOUS FILENAMES NOT ALLOWED'
	DB	CR,LF,'\$'
COLONERR	DB	CR,LF,'COLON (:) NOT PROPERLY PLACED IN
	DB	'FILENAME',CR,LF,'\$'
FNCHARER	DB	CR,LF,'FILENAME TOO LONG OR TOO SHORT'
	DB	CR,LF,'(8 CHARS MAX, 1 CHAR MIN)',CR,LF,'\$'
HEXFTERR	DB	CR,LF,'HEX FILETYPES ONLY !',CR,LF,'\$'
SPFNERR	DB	CR,LF,'NO SPACES ALLOWED IN FILENAME'
	DB	CR,LF,'\$'
NPRTERR	DB	CR,LF,'NON-PRINTABLE CHARACTERS NOT
	DB	'ALLOWED IN FILENAME',CR,LF,'\$'
FNFNERR	DB	CR,LF,'FILE NOT FOUND',CR,LF,'\$'
CKSUMERR	DB	CR,LF,'HEX CHECKSUM ERROR',CR,LF,'\$'
DSKRDERR	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
	DB	'THAN FINISH ADDRESS',CR,LF,'\$'
DDSPCERR	DB	CR,LF,'OUT OF DIRECTORY OR DISK STORAGE
	DB	'SPACE',CR,LF,'PARTIAL FILE WAS NOT
	DB	'SAVED !',CR,LF,'\$'
DRIVERR	DB	CR,LF,'WARNING - ONLY CURRENTLY SELECTED
	DB	'DISK WILL BE USED, INPUT IGNORED !'
	DB	CR,LF,'\$'
CNTRLMSG	DB	CR,LF,'MDS IS IN CONTROL, CAN'T CONTINUE'
	DB	'UNTIL OPTION 'E' IS SELECTED',CR,LF,'\$'
ABORTMSG	DB	CR,LF,'ABORT MDS CONTROL (Y/N)? \$'
ABORTEDM	DB	CR,LF,'MDS CONTROL ABORTED, HOST IN
	DB	'CONTROL.',CR,LF,'\$'
EXMSG	DB	CR,LF,'WILL CONSOLE BE RECEIVING DATA
	DB	'FOR DISPLAY FROM THE MDS (Y/N)?\$'
EXMSG2	DB	CR,LF,LF
	DB	'MDS IS IN CONTROL, HOST MAY REGAIN
	DB	'CONTROL ONLY BY TYPING THE ESCAPE KEY !'
	DB	CR,LF,LF,'\$'
FORMTMSG	DB	CR,LF,'INPUT PARAMETER FORMATS ARE AS
	DB	'FOLLOWS :',CR,LF
	DB	'MENU >X
	DB	'X IS OPTION SELECTION (A-N)',CR,LF
	DB	'HEXARITH >XXXX YYYY
	DB	'XXXX & YYYY ARE HEX INTEGERS',CR,LF
	DB	'DOWNLOAD >FILENAME(.HEX)
	DB	'(.HEX) IS OPTIONAL',CR,LF
	DB	'UPLOAD >FILENAME(.HEX)',CR,LF
	DB	'>XXXX YYYY


```

      LB      '   XXXX IS MDS HEX ADDRESS WHERE EXECUTION '
      DB      CR,LF
      LB      '
      LB      '   IS TO BEGIN',CR,LF,LF
      DB      'PRESS ANY KEY TO CONTINUE,>$'
HEXMSG      LB      CR,LF,'HEX ADD/SUB',CR,LF,'$'
HEXMSG1     LB      'SUM = $$$$'
HEXMSG2     DB      'DIFF = $$$$',CR,LF,'$'
EXAMSG      LB      CR,LF,'EXAMINE/SET MDS MEMORY',CR,LF,'$'
EXAMSG1     LB      '$$$$'
EXAMSG2     DB      '$$ $'
FILLMSG     LB      CR,LF,'FILL MDS MEMORY LOCATION(S)',CR,LF
            LB      '$'
CSETMSG     DB      CR,LF,'CONTINUOUS SET MDS MEMORY W/O '
            LB      'EXAMINE',CR,LF,'$'
EXECMSG     LB      CR,LF,'EXECUTE MDS MEMORY FROM SPECIFIED '
            DB      'ADDRESS',CR,LF,'$'
LOCMSG      LB      CR,LF,'LOCATE BYTE SEQUENCE IN MDS MEMORY'
            LB      CR,LF,'$'
NOTFOUND    DB      CR,LF,'BYTE SEQUENCE NOT FOUND !',CR,LF,'$'
FOUNIMSG    LB      CR,LF,'FOUND STARTING AT MDS ADDRESS '
FOUNIMSG1   LB      '$$$$ ',CR,LF,'$'
DUMPMMSG    DB      CR,LF,'DUMP MDS MEMORY',CR,LF,'$'
DUMPMMSG1   LB      '$$$$ $'
DUMPMMSG2   LB      '$$ $'
DUMPMMSG3   DB      '$$$$$$$$$$$$$$$$$$'
MENUPRO1    LB      CR,LF,'OPTION A = MENU SUPPRESSION, B = '
            LB      'NO MENU SUPPRESSION'
MENUPROM    DB      CR,LF,'INPUT MENU OPTION $'
PROMPT      LB      '>$'
FILENAME    LB      'FILENAME $'
DWNLDMSG    DB      CR,LF,'DOWNLOAD HEX FILE FROM DISK TO MDS'
            LB      'MEMORY',CR,LF,'$'
DWNLDONE    LB      CR,LF,'DOWNLOAD COMPLETED',CR,LF
DWNLDONE1   DB      'MDS START ADDRESS = $$$$H , LAST ADDRESS '
            LB      '= $$$$H',CR,LF,'$'
UPLDMSG     LB      CR,LF,'UPLOAD (SAVE) MDS MEMORY TO DISK '
            DB      'HEX FILE',CR,LF,'$'
UPLDONE     LB      CR,LF,'UPLOAD TO DISK SUCCESSFULLY '
            LB      'COMPLETED',CR,LF,'$'
DATAMSG     DB      '$$ $'
STARMSG     LB      '*$'
FCBMSG      LB      0,20H,20H,20H,20H,20H,20H,20H
            DB      'HEX',0,0,0,0

```

```

SYSSTAT     IS      1          ;SYSTEM STATUS FLAG
                                ; HOST IN CONTRCL = 0
                                ; MDS IN CONTROL = 1
MENUSUPF    IS      1          ;MENU SUPPRESSION FLAG
                                ; 0 = NC SUPPRESSION

```

MENUFLG	DS	1	; 1 = SUPPRESSION
FRSTDEL	DB	0	; STORAGE FOR MENU CHOICE
FIRST	DW	0	; FIRST DELIMITER FLAG
SECOND	DW	0	; FIRST NUMBER TO ADD/SUB
SUM	DW	0	; SECOND NUMBER TO ADD/SUB
START	DW	0	; SUM OF HEX NUMBERS
			; STARTING ADDRESS FOR
			; COMMAND USE
FINISH	DW	0	; FINISH ADDRESS FOR
			; COMMAND USE
MISDATA	DS	1	; TEMP. STORAGE FOR DATA
			; FROM MIS
CONSDATA	DS	1	; TEMP. STORAGE FOR DATA
			; FROM CONSOLE TO MIS
MDSRDYF	DS	1	; MIS READY FLAG
			; 0FFH = DONE, 0 = NOT DONE
FIRSTIME	DS	1	; FIRST TIME THROUGH READ
BUFCNT	DS	1	; BUFFER COUNT SPACE
CURRENT	DS	1	; CURRENT DISK DRIVE
CONTF LG	DS	1	; CONTINUATION FLAG FOR DISK
			; READ OPERATIONS
			; 00 = NO 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
END		STARTER	

APPENDIX C

MDS MONITOR SOFTWARE LISTING

```

*****
*
*      AMIS1 - ALTOS MICROCOMPUTER DEVELOPMENT SYSTEM
*              (MIS CODE)
*
* VERSION 1.3,  28 MAY 1981
* LT. STEPHEN M. HUGHES - AUTHOR
*
*      THIS IS THE MIS MONITOR CODE 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
CHBSTAT EQU 0E2H ;CHANNEL B STATUS AND
; COMMAND/CONTROL PORT
CHBDATA EQU 0E1H ;CHANNEL B DATA PORT
; (NOT USED IN TEIS 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
```

	XRA	A	
	STA	OPTION	;RESET OPTION FLAG
	CALL	HOSTIN	;GET COMMAND FROM HOST
MONITOR1	ANI	7FH	;COMMAND WILL BE ASCII
	CPI	'W'	;DOWNLOAD COMMAND?
	JZ	DWNLD	
	CPI	'U'	;UPLOAD COMMAND?
	JZ	UPLD	
	CPI	'X'	;EXAMINE/SET MEMORY CMT?
	JZ	EXAM	
	CPI	'C'	;CONTINUOUS MEMORY SET CMT?
	JZ	CSET	
	CPI	'F'	;FILL COMMAND?
	JZ	FILL	
	CPI	'L'	;LOCATE SEQ. COMMAND?
	JZ	LOCATE	
	CPI	'D'	;DUMP MEMORY COMMAND?
	JZ	DUMP	
	CPI	'E'	;EXECUTE MEMORY CMT?
	JZ	EXEC	
	JMP	MONITOR	;ANYTHING ELSE IS IGNORED

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

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

* UPLD - UPLOAD MDS MEMORY TO DISK HEX FILE *

UPLD	CALL	GETADDR	;GET STARTING ADDRESS
	SHLD	START	
	CALL	GETADDR	;GET FINISH ADDRESS
	SHLD	FINISH	
	LHLD	START	
	XCHG		;DE = START ADDRESS
UPLD1	LDAX	D	;GET DATA
	CALL	HDATAOUT	;SEND IT
	INX	D	
	CALL	BUFFCMP	;DONE YET?
	RRC		
	JNC	UPLD1	; NO

```

CALL    MISRIY      ; YES
JMP     MONITOR

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

EXAM    CALL    GETADDR      ;GET STARTING ADDRESS
EXAM1   MOV     A,M          ;SEND DATA AT HL ADDRESS
                                   ; TO HOST

        CALL    HDATAOUT
        CALL    HOSTIN       ;GET NEW DATA
        MOV     M,A          ; DEPOSIT IT
        INX     H
        JMP     EXAM1        ;LOOP TILL HOSTDONE

* CSET - CONTINUOUS SET OF MIS 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
        LHL    START
        XCHG
        MOV     A,C          ;LE = START ADDRESS
FILL1   STAX    D             ;GET FILL DATA
        INX     D             ; DEPOSIT IT
        CALL    BUFCMP       ;DONE YET?
        RRC
        JNC     FILL1        ; NO, KEEP FILLING
        CALL    MISDONE      ; YES
        JMP     MONITOR

* LOCATE - LOCATE BYTE SEQUENCE IN MIS 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
LOCIN	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
	LHLD	START	
	XCHG		;DE = START ADDRESS
	LXI	H,DATABUFF	;HL = START OF SEQUENCE
SRCH1	LDAX	D	;GET MDS DATA
	CMP	M	; IS THERE A MATCH?
	JZ	MATCH	; YES
	INX	D	
	CALL	BUFFCMP	; NO, SEE IF DONE
	RRC		
	JC	NOTFND	;YES, SEQ. NOT FOUND
	JMP	SRCH1	;NO, TRY AGAIN
MATCH	XCHG		;HL = FIRST MATCH ADDRESS
	SHLD	MATCHADR	; SAVE IT
	XCHG		;RESTORE DE & HL
MATCH1	DCR	C	;ALL MATCHES YET?
	JZ	FOUND	; YES, FOUND SEQUENCE
	INX	D	
	CALL	BUFFCMP	;DONE YET?
	RRC		
	JC	NOTFND	; YES, SEQ. NOT FOUND
	INX	H	; NO, LOOK FOR NEXT MATCH
	LDAX	D	
	CMP	M	;ANOTHER MATCH?
	JZ	MATCH1	; YES
	LHLD	DATABUFF	; NO, START ALL OVER
	INX	D	
	LDA	LOCOUNT	;RE-INIT. SEC. COUNT
	MOV	C,A	
	JMP	SRCH1	;KEEP TRYING
FOUND	MVI	A,'F'	;SEND FOUND TO HOST
	CALL	HDATAOUT	
	LHLD	MATCHADR	;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 MDS MEMORY TO HOST CONSOLE *

```

DUMP    CALL  GETADDR          ;GET START ADDRESS
        SHLD  START
        CALL  GETADDR          ;GET FINISH ADDRESS
        SHLD  FINISH
        LHL   START
        XCHG                      ;DE = START ADDRESS
DUMP1   LDAX  D                 ;GET MDS MEMORY DATA
        CALL  HDATAOUT
        INX   D
        CALL  BUFCMP           ;DONE YET?
        RRC
        JNC   DUMP1            ; NO
        CALL  MDSRY            ; YES
        JMP   MONITOR

```

```

* EXEC - EXECUTE MDS MEMORY
*       PROGRAM TO BE EXECUTED MAY RETURN MONITOR VIA
*       A 'RST ?' 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 ***

```

* BUFCMP - COMPARE DE TO FINISH ADDRESS + 1
*         IF EQUAL, RETURN A = 0FFH
*         IF UNEQUAL, RETURN A = 00 *

```

```

BUFCMP  PUSH  H
        PUSH  D                 ;DE=CURRENT ADDR TO COMPARE
        LHL   FINISH           ;HL = FINISH ADDRESS + 1
        INX   H
        MOV   A,H               ;H = D?
        CMP   L
        JNZ   NOCMP             ; NO
        MOV   A,L               ; YES, L = E?
        CMP   E
        JNZ   NOCMP             ; NO
        MVI   A,0FFH           ; YES, ADDRESSES ARE EQUAL
        POP   L

```

```

      POP      H
      RET
NOCMP  XRA      A          ;ADDRESSES NOT EQUAL
      POP      C
      POP      H
      RET

```

* GETADDR - GET ADDRESS FROM HOST *

```

GETADDR CALL  HOSTIN      ;GET MSB FIRST
      MOV     H,A
      CALL    HOSTIN      ; THEN LSB
      MOV     L,A
      RET

```

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

```

PORTSET MVI     A,77H      ;SET RATE TO 9600 BAUD
      OUT      BAUDREG
      MVI     A,01001110B  ;SEND CONTROL BYTE
                          ; 1 STOP BIT
      OUT      CHASTAT     ; NO PARITY, 8 BITS/CHAR
      OUT      CHSTAT      ; 16x RATE FACTOR
      MVI     A,00110111B  ;SEND COMMAND BYTE
      OUT      CHASTAT
      OUT      CHSTAT
      JMP     MONITOR

```

* USRIO - USER TO/FROM HOST CONSOLE I/O ROUTINE
* USER EXECUTED PROGRAMS IN MMS 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 AND REG. C = 2
* - TO CHECK THE FOR HOST INPUT, CALL WITH
* REG. C = 3 - RETURNS A = 00 IF NO INPUT HAS BEEN
* RECEIVED FROM THE HOST; A = 0FFH IF INPUT IS
* WAITING
* - IF C <> 1, 2 or 3 THEN ROUTINE RETURNS WITH C = 0FFH

```

USRIO  PUSH     PSW
      MOV     A,C          ;SEE IF INPUT OR OUTPUT
      CPI     1
      JZ      USRIN
      CPI     2
      JZ      USROUT
      CPI     3          ;WANT STATUS ?
      CZ      HOSTAT      ; YES, GET IT

```

```

                MVI      C,0FFH      ;ILLEGAL CODE
                RET
USRIN          CALL     MISRDY      ;TELL HOST TO SEND INPUT
                POP      PSW
                CALL     HOSTIN      ;GET INPUT
                RET          ; RETURN WITH IT IN A
USROUT         POP      PSW
                CALL     H1ATAOUT    ;SEND CHARACTER TO HOST
                RET

```

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

```

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

```

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

```

HOSTIN  CALL     GETCHAR      ;GET INPUT
HOSTIN1 CPI      55H          ;IS IT A COMMAND?
        JZ       HOSTCMD
        CPI      0FFH        ;IS IT DATA?
        JZ       HOSTDTA
        JMP      HOSTRDY     ;MUST BE HOST READY FLAG
HOSTCMD 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,0FFH      ; SET FLAG IN MDS
        STA      HSTRDYFL
        RET          ;RETURN TO CALLER

GETCHAR CALL     HOSTAT       ;LOOP TILL CHAR. IS WAITING
        RRC
        JNC      GETCHAR1
GETCHAR1 IN      CHADATA      ;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                ; HAS PRIORITY)
JNC                ; NO
CALL GETCHAR1      ; YES, GET IT
CALL HOSTIN1       ; IF COMMAND, BACK TO MONITOR
                   ; ELSE IGNORE IT
                   ; GET PORT STATUS
HCSTOUT1 IN        CHASTAT
ANI 1
JZ HOSTOUT1        ; LOOP TILL READY TO SENI
POP PSW            ; SEND CHARACTER
OUT CHADATA
CPI XON            ; DON'T WAIT FOR XON
RZ                ; CONFIRMATION
XONCK CALL HOSTAT  ; WAIT FOR CONFIRMATION
RRC
JNC XONCK
IN CHADATA         ; GET IT
RET

```

* HOSTAT - HOST INPUT STATUS *

```

HOSTAT IN          CHASTAT
ANI 2
RZ                ; NO CHAR. WAITING, RET A=0
MVI A,0FFH        ; CHAR. WAITING, RET A=0FFH
RET

```

* HDATAOUT - SEND DATA TO HOST IN PROPER FORMAT *

```

HDATAOUT 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

```

RET

*** DATA STORAGE AREAS - IN ONBOARD RAM ***

	ORG	RAM	
HSTRLYFL	DS	1	;HOST READY FLAG
			; 00 = NOT READY
			; 0FFH = READY
MATCHADR	DW	0	;STORAGE FOR FIRST ADDRESS
			; OF MATCH
LOCOUNT	DS	1	;STORAGE FOR BYTE COUNT
START	DW	0	;STORAGE FOR START &
FINISH	DW	0	; FINISH ADDRESSES
OPTION	DS	1	;STORAGE FOR OPTION SELECTED
	DS	63	;ALLOW FOR A 32 LEVEL STACK
STACK	DS	1	
DATABUFF	DS	25	;STORAGE FOR LOCATE SEQUENCE

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. LOBB'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
BKSPACE 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
WCNT    EQU      3              ;SEQUENTIAL WRITES

MEM      DI                      ;DISABLE INTERRUPTS
        LXI      SP,STACK        ;INITIALIZE STACK
        LXI      B,TEND          ;FORMAT ADDRESS OF END OF TEST
        LXI      H,MEMT1
        CALL     CHA

* TEST STARTS HERE *

MEM01   CALL     CRLF            ;MAKE OUTPUT PRETTY
        LXI      H,0000H         ;INITIALIZE PAS COUNT,
                                   ; CUMULATIVE ERROR COUNT
```

	SHLD	MEMF	; AND ADDRESS 'OR' PRODUCT
	SHLD	MEMX	
	SHLD	MEML	
	LXI	H,-1	; INIT. ADDRESS 'AND'
	SHLD	MEMK	
	LXI	H, MEMA	; PRINT PROGRAM TITLE
	CALL	DSPLY	
* GET TEST MODE *			
MEM03	MVI	A,1	; SET DEFAULT = ITEMIZE
	STA	MEMP	
	LXI	H, MEMN	
	CALL	DSPLY	; PRINT SELECT I, T OR E
	CALL	CRLF	
	MVI	A, '>'	; PROVIDE A CUE MARK
	CALL	USROUT	
	CALL	USRIN	; WAIT FOR INPUT
	ORI	20H	; MAKE LOWER CASE
	CPI	'e'	; IF E, EXIT
	JZ	MEM55	
	CPI	'i'	; IF I, ITEMIZE ERRORS
	JZ	MEM04	
	CPI	't'	; IF T, PRINT TOTAL ERRORS
			; ONLY
	JNZ	MEM03	; IF NONE, TRY AGAIN
	XRA	A	; SET TOTAL ONLY FLAG
	STA	MEMP	
* GET MEMORY TEST LIMITS *			
MEM04	LXI	H, MEMB	; PRINT ENTER FBA
	CALL	DSPLY	
	CALL	ENTR	; GET 16 BIT ADDRESS
	MOV	A, H	; IF UPPER BYTE OF FBA IS
	ORA	A	; NEGATIVE, OK TO USE
	JM	MEM05	; SO JUMP
	LXI	D, TEND	; OTHERWISE, MAKE SURE FBA
	PUSH	H	; IS NOT WITHIN TEST PROGRAM
			; AREA
	MOV	A, L	; (HL = HL - DE - C)
	SUB	E	
	MOV	L, A	
	MOV	A, H	
	SBB	C	
	MOV	H, A	
	POP	H	
	JP	MEM05	; FBA IS OK, JUMP
MEM045	LXI	H, MEMT	; IF FBA IS WITHIN TEST PROGRAM
	CALL	DSPLY	; AREA, SET IT TO END OF

```

MEM05    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
        ORA      A           ;CLEAR CARRY FLAG
        PUSH     H           ; (DE = CONTENTS OF MEMI
                                   ;      AND MEMI + 1)

        LHLD     MEMI
        MOV      D,H
        MOV      E,L
        POP      H
        MOV      A,L        ;MAKE SURE FBA < LBA
        SUB      E          ; (HL = HL - DE - C)
        MOV      L,A
        MOV      A,H
        SBB      D
        MOV      H,A
        JNC      MEM06      ;IT'S OK, JUMP
        POP      H          ;RESTORE STACK
        POP      H
        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
        LHLD     MEMI        ; (BC = CONTENTS OF MEMI
                                   ;      AND MEMI + 1)

        MOV      B,H
        MOV      C,L
        POP      H
        LXI      H,MEMG
        CALL     CHA
        POP      H           ;HL = LBA
        PUSH     H

MEM08    LXI      H,MEMV      ;PRINT ABORT INSTRUCTION
        CALL     DSPLY
        POP      D           ;DE = LBA
        INX      D          ;LBA = LBA + 1

```

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

```

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   MEM1         ;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
        MOV    M,A          ; ...WRITE IT
        INX    H            ;ADVANCE MEMORY ADDRESS
        MOV    A,L          ;CHECK IF END OF AREA TO BE
        CMP    E            ; TESTED
        JNZ    MEM21        ;LOOP, NOT YET
        MOV    A,H
        CMP    D
        JNZ    MEM21        ;LOOP, NOT DONE YET
        POP    B            ;GET WRITES COUNTER
        ECR    B            ;WRITE PATTERN OVER AND OVER
        JNZ    MEM2
        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   MEM1         ;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
        MOV    B,A          ;...SAVE IT
        MOV    A,M          ;READ MEMORY
        CMP    B            ;IS DATA CORRECT?
        JZ     MEM32        ; YES, JUMP
        MOV    M,B          ;WRITE THE CORRECT DATA

```


MEM5	CALL	USRIN	;GET INPUT
	CPI	04H	;D - FREEZE ACTION
	JZ	DISPSTP	
	ORI	20H	;FOLD TO LOWER CASE
	CPI	'1'	;DYNAMIC SET ITEMIZE
	JZ	MAKEI	
	CPI	't'	;DYNAMIC SET TOTAL ONLY
	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		;DON'T PRINT IT
STACKIT	LXI	SP, STACK	;RESET STACK
	JMP	MEM01	;RESTART TEST
MAKEI	MVI	A, 1	;MAKE ITEMIZE
	STA	MEMF	
	CALL	BSOUT	
	RET		
MAKET	MVI	A, 0	;MAKE TOTAL ONLY
	STA	MEMF	
	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
			; AND MEMF + 1)
	LHLD	MEMF	
	MOV	B, H	
	MOV	C, L	
	POP	H	
	INX	B	;COUNT PASSES
	PUSH	H	; (MOV BC TO MEMF)
	MOV	H, B	
	MOV	L, C	

SHLD	MEMF	
POP	H	
LXI	H, MEMG1	; CONVERT PASS COUNT
CALL	CHA	
PUSH	H	; (BC = CONTENTS OF MEME
		; AND MEME + 1)
LHLD	MEME	
MOV	B, H	
MOV	C, L	
POP	H	
LXI	H, MEMG2	; CONVERT ERROR COUNT
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	B	
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	; MAKE SURE NO ERRORS
ORA	L	
JZ	MEMG7	; NONE YET, JUMP
MVI	A, ' '	; REMOVE THE CARRIAGE RETURN
		; FROM THE OUTPUT STRING
STA	MEMG25	
PUSH	H	; (BC = CONTENTS OF MEMK
		; AND MEMK + 1)
LHLD	MEMK	
MOV	B, 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	

```

                MOV    C,L
                POP    H
                LXI    H,MEMG4          ;CONVERT LOGICAL 'OR' OF
                                         ; FAILING ADDRESSES
MEM67          CALL    CHA
                LXI    H,MEMG          ;PRINT PASS AUDIT
                CALL    DSPLY
                LDA    MEMJ            ;ROTATE BIT CROSSTALK SO THAT
                RLC                     ; OVER EIGHT PASSES ALL BIT
                STA    MEMJ            ; PATTERNS WILL BE USED
                POP    D                ;RESTORE LBA+1
                JMP    MEM1            ;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,'C'
                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, MEMD4	; CONVERT 'OR' FOR OUTPUT
	CALL	CHAB	
	POP	PSW	; GET MEMORY CONTENTS AND
			; CONVERT IT FOR OUTPUT
	MOV	C, A	
	LXI	H, MEMD3	
	CALL	CHAB	
	MOV	C, B	; CONVERT PATTERN
	LXI	H, MEMD2	
	CALL	CHAB	
	POP	B	; CONVERT CURRENT MEMORY ADDRESS
	PUSH	B	
	LXI	H, MEMD1	
	CALL	CHA	
	LHLD	MEME	
	INX	H	; COUNT ERRORS THIS PASS
	SHLD	MEME	
	POP	D	; GET CURRENT MEMORY ADDRESS
	PUSH	D	
	LHLD	MEMK	
	MOV	A, D	; SAVE LOGICAL 'AND' OF
			; FAILING ADDRESSES
	ANA	H	
	MOV	H, A	
	MOV	A, E	
	ANA	L	
	MOV	L, A	
	SHLD	MEMK	
	LHLD	MEML	
	MOV	A, D	; SAVE LOGICAL 'OR' OF
			; FAILING ADDRESSES
	ORA	H	
	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

```

*
*
* RETURN      C = PATTERN COUNTER
               A = DATA PATTERN *

PATTN  PUSH    H                ;PATTERN COMPUTATION
        MVI    B,0              ;BRANCH ON PATTERN
        LXI    H,PATT0-3
        DAD    B
        DAD    B
        DAD    B
        KTHL                ;(RESTORE MEM ADDR)
        NOP
        RET                ;(BRANCH)
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    B
        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
        ADI     7                ;YES
CHA15   ADI     '0'              ;FORM ASCII
        MOV     M,A              ;STORE THIS CHARACTER
        DCX     H                ;BACK UP THROUGH OUTPUT AREA
        MVI     E,4              ;DOUBLE RIGHT

```

```

CHA2    ORA      A                ;SHIFT 4 BITS
        MOV      A,B
        RAR
        MOV      B,A
        MOV      A,C
        RAR
        MOV      C,A
        DCR      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,DE,BC UNCHANGED
*           A DESTROYED *

```

```

CHAB     PUSH     H                ;SAVE REGISTERS
        PUSH     D
        PUSH     B
        INX      H
        MVI      D,2
        JMP      CHA1

```

* 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?
        RZ                     ;YES, EXIT
        CPI    LF         ;LINE FEED?
        RZ                     ;YES, EXIT
        CPI    'A'        ;IS IT 0-9?
        JC     ENTR15     ;YES
        ANI    0DFH       ;NO, FORCE LOWER CASE
ENTR15   DAD    H          ;SHIFT PREVIOUS DATA LEFT
        DAD    H          ; 4 BITS
        DAD    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     ENTR2      ;NO, IT'S 0-9
        ADI    9          ;ADD FUDGE FACTOR
ENTR2    ANI    0FH        ;ISOLATE 4 BITS
        ORA    L          ;MERGE WITH PREVIOUS DATA
        MOV    L,A
        DCR    C          ;COUNT CHARACTERS
        RZ                     ;EXIT IF 4 RECEIVED
        JMP    ENTR1      ;GET ANOTHER CHARACTER
ENTR3    MVI    A,'?'     ;PRINT QUESTION MARK
        CALL   USROUT
        JMP    ENTR       ; 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'
        DB     'TO TEST:',CR

```

```

MEMC    DB      'ENTER ADDRESS OF LAST MEMORY BYTE'
        DB      ' TO TEST:',CR
MEMD    DB      'ADDRESS='
MEMD1   DB      '$$$$ PATTERN='
MEMD2   DB      '$$ CONTENTS='
MEMD3   DB      '$$ XOR='
MEMD4   DB      '$$ TYPE='
MEMD5   DB      ',CR
MEME    DW      0          ;ERRORS THIS PASS
MEMF    LW      0          ;PASS COUNT
MEMG    DB      '$$$$-$$$$ PASS: '
MEMG1   DB      '$$$$ ERRORS: '
MEMG2   DB      '$$$$ CUM. ERRORS: '
MEMG23  DB      '$$$$ '
MEMG25  DB      CR, 'AND: '
MEMG3   DB      '$$$$ OR: '
MEMG4   DB      '$$$$ ',CR
MEMI    DW      0          ;FIRST BYTE ADDRESS TO TEST
MEMJ    LB      0FEH      ;BIT CROSSTALK PATTERN
MEMK    LW      -1        ;LOGICAL 'AND' OF FAILING
        ; ADDRESSES
MEML    LW      0          ;LOGICAL 'OR' OF FAILING
        ; ADDRESSES
MEMM    DB      LF, 'GOODBYE',CR
MEMN    DB      'I=ITEMIZE ERRORS, '
        DB      'T=PRINT ERROR TOTAL ONLY, '
        DB      'E=EXIT TEST',CR
MEMP    LB      0          ;FLAG 1=ITEMIZE, 0=TOTAL
MEMT    LB      'END OF PROGRAM USED AS FIRST '
        DB      'ADDRESS TO TEST = '
MEMT1   LB      '$$$$ ',CR
MEMU    DB      'ERROR: LAST BYTE ADDRESS LESS '
        DB      'THAN FIRST BYTE ADDRESS.',CR
MEMV    LB      LF
        DB      'TO ABORT TEST PUSH ANY KEY'
        DB      CR
MEMX    LW      0          ;CUMULATIVE ERROR COUNT

USRIN   PUSH    B          ;GET INPUT FROM HOST CONSOLE
        PUSH    C
        PUSH    H
        MVI     C,1
        CALL    USRIO
        POP     H
        POP     D
        POP     B
        RET

USROUT  PUSH    B          ;SEND CHARACTER TO HOST
        PUSH    C          ; CONSOLE

```

	PUSH	H	
	MVI	C,2	
	CALL	USRIO	
	POP	H	
	POP	D	
	POP	B	
	RET		
USRSTAT	PUSH	B	;SEE IF CHARACTER IS WAITING
	PUSH	C	
	PUSH	H	
	MVI	C,3	
	CALL	USRIO	
	POP	H	
	POP	D	
	POP	B	
	RET		
BSOUT	MVI	A,BKSPACE	;PRINT A BACKSPACE
	CALL	USROUT	
	RET		
STACK	DS	64	;SET UP FOR 32 LEVELS
TEND	EQU	\$+2	
	END	100H	

APPENDIX F

SAMPLE MENU LISTING

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

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 XXXX & YYYY ARE HEX INTEGERS
 DOWNLOAD >FILENAME(.HEX) (.HEX) IS OPTIONAL
 UPLOAD >FILENAME(.HEX)
 >XXXX YYYY
 EXAMINE MDS >XXXX XXXX & YYYY ARE MDS HEX START AND
 END ADDRESSES FOR UPLOAD
 XXXX IS FIRST MDS HEX ADDRESS TO
 EXAMINE AND SET
 XXXX IS HEX ADDRESS, YY 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 XXXX & 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 MDS HEX START AND OPTIONAL END ADDRESSES TO SEARCH BETWEEN
	>AA BB ... PP	ARE UP TO A 16 BYTE HEX SEQUENCE
DUMP	>XXXX(YYYY)	TO SEARCH FOR IN MDS MEMORY
	>XXXX	XXXX & YYYY ARE MDS HEX START AND OPTIONAL END ADDRESSES TO DUMP BETWEEN
EXECUTE		XXXX IS MDS HEX ADDRESS WHERE EXECUTION IS TO BEGIN

PRESS ANY KEY TO CONTINUE >

BIBLIOGRAPHY

Barden, William Jr., The Z80 Microcomputer Handbook, Howard W. Sams & Co., Inc., 1979.

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

PRO-LOG CORPORATION, 7304 Dual Uart Card Users Manual, 1980.

PRO-LOG CORPORATION, 7701 16K Static Memory Card Users Manual, 1980.

PRO-LOG CORPORATION, 7803 Processor Card (Z80) Users Manual, 1980.

PRO-LOG CORPORATION, Series 7000 STD BUS Technical Manual and Product Catalog, March 1981.

Titus, Jonathan A. and others, The 8080A Bugbook, 1st ed., Howard W. Sams & Co., Inc., 1977.

Titus, Jonathan A. and others, 8080/8085 Software Design - Book 1, 1st ed., Howard W. Sams & Co., Inc., 1980.

Titus, Jonathan A. and others, 8080/8085 Software Design - Book 2, 1st ed., Howard W. Sams & Co., Inc., 1979.

Titus, Jonathan A. and others, Interfacing and Scientific Data Communications Experiments, 1st ed., Howard W. Sams & Co., Inc., 1980.

Zaks, Rodney, How to Program the Z80, 3rd ed., SYBEX Inc., 1979.

INITIAL DISTRIBUTION LIST

	No. Copies
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 93940	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

