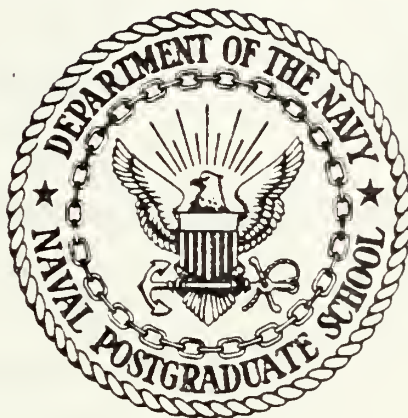
THE IMPLEMENTATION OF AN OPERATING SYSTEM FOR A SHARED MICROCOMPUTER ENVIRONMENT

Stephen J. Carro

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

The Implementation of an
Operating System for a
Shared Microcomputer Environment

by

Stephen J. Carro
and
Barry L. Knouse

September 1977

Thesis Advisor: G. A. Kildall

Approved for public release; distribution unlimited.

T180628

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The Implementation of an Operating System for a Shared Microcomputer Environment		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; September 1977 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Stephen J. Carro Barry L. Knouse		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE September 1977 13. NUMBER OF PAGES 198
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 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)

microcomputer
operating system
resource sharing
microprocessor
timesharing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An implementation of a multiuser timeshared operating system for the Sycor 440 Clustered Terminal Processing System has been described. Utilizing an 8080 microprocessor, this system provides a virtual operating environment consisting of a console device, eight floppy disk drives, and up to 32 virtual floppy disk images on a five megabyte movable-head disk. In

DD FORM 1473, 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

addition a Centronix serial printer is incorporated into the system as a dedicated device for any one of up to four concurrent users. The operating system supports utility programs including an editor, assembler, and debugger which facilitate microcomputer program development at the Naval Postgraduate School.

Approved for public release; distribution unlimited.

The Implementation of an Operating System for a Shared Microcomputer Environment

by

Stephen J. Carro
Lieutenant, United States Navy
B.S., United States Naval Academy, 1971

and

Barry L. Knouse
B.S., Pennsylvania State University, 1972

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September, 1977

ABSTRACT

An implementation of a multiuser timeshared operating system for the Sycor 440 Clustered Terminal Processing System is described. Utilizing an 8080 microprocessor, this system provides a virtual operating environment consisting of a console device, eight floppy disk drives, and up to 32 virtual floppy disk images on a five megabyte movable-head disk. In addition, a Centronix serial printer is incorporated into the system as a dedicated device for any one of up to four concurrent users. The operating system supports utility programs including an editor, assembler, and debugger which facilitate microcomputer program development at the Naval Postgraduate School.

TABLE OF CONTENTS

TABLE OF ABBREVIATIONS.............................. 7

I. INTRODUCTION...................................... 8

A. BACKGROUND..................................... 8

B. GOALS AND OBJECTIVES........................... 9

C. PROBLEM DEFINITION............................. 10

II. SYCOR 440 HARDWARE DESCRIPTION.................... 11

III. MIS DESCRIPTION.................................. 14

A. BACKGROUND..................................... 14

B. MIS INTERNAL DESIGN FEATURES................... 14

IV. SYSTEM DEBUGGING TOOLS............................ 18

A. SYCOR HARDWARE................................. 18

B. MIS DEBUGGER TOOL (MDT)........................ 19

V. MIS ENHANCEMENT................................... 20

A. FILE INTEGRITY................................. 20

B. SYSTEM LOAD PROCEDURE.......................... 20

C. PRINTER INTEGRATION............................ 21

VI. IMPLEMENTATION OF AN OPERATING SYSTEM............. 24

A. OPERATING SYSTEM SELECTION..................... 24

B. OPERATING SYSTEM IMPLEMENTATION................ 24

VII. CONCLUSIONS AND RECOMMENDATIONS.................. 27

APPENDIX A MIS USER'S MANUAL.......................... 29

MTS PROGRAM LISTINGS....................................104

LIST OF REFERENCES....................................196

INITIAL DISTRIBUTION LIST....................................198

TABLE OF ABBREVIATIONS

ASCII	- American Standard Code for Information Interchange
BDOS	- Basic Disk Operating System
CCP	- Command Control Program
CP/M	- Control Program for Microcomputers
CPU	- central processing unit
CRT	- cathode ray tube
DMA	- direct memory access
H	- hexadecimal (base 16 number system)
I/O	- input/output
IPL	- initial program load
K	- tenth power of 2, i.e. 1024
LSI	- large scale integration
MDT	- MTS Debugger Tool
MTS	- Microcomputer Timeshared System
NPS	- Naval Postgraduate School, Monterey, Calif.
PL/M	- Programming Language for Microcomputers
RAM	- random access memory
ROM	- read only memory
SPOOL	- simultaneous peripheral operations on line
TPA	- transient program area
TTY	- teletypewriter
VMM	- virtual machine monitor

I. INTRODUCTION

A. BACKGROUND

In the summer of 1976 the Computer Science Department at the Naval Postgraduate School (NPS) acquired a Sycor Model 440 Clustered Terminal Processing System for use in the NPS microcomputer laboratory. The Sycor 440 utilizes an Intel 8080 LSI chip as the CPU of a special purpose microcomputer system designed primarily for data entry applications. In addition to the 8080 CPU, the Sycor system hardware configuration includes a five megabyte movable-head disk, 64K of random access memory (RAM), a cassette tape drive, and four Sycor 4412 display terminals consisting of a keyboard and cathode ray tube (CRT) display device. Other system devices include a model 340 communications terminal, a Centronix serial printer and an RS-232 asynchronous communication interface. As part of the Model 440 System, Sycor supplied a comprehensive package of system software. Most of this software was designed to support applications in the data entry field which comprised the Sycor 440's primary market.

While the Sycor 440 made a significant addition to the hardware complement of the NPS microcomputer laboratory, a great deal of effort was needed to integrate the system into the microcomputer laboratory to ensure that maximum benefit

would be derived from the new equipment. The first step in the integration of the Sycor 440 hardware within the microcomputer laboratory was undertaken as a thesis project in the winter of 1977. Kenneth J. Brown and David R. Bullock deserve a great deal of credit for the conception and the design of the Microcomputer Timeshared System (MTS). Their thesis work is contained in Ref. 1. MTS was developed as a virtual machine monitor (VMM) to support a user operating system for microcomputer program development. An in depth description of the system is given in Chapter III. Because much of MTS had not been debugged and because it did not support an operating system which would accommodate student use, thesis work was continued in the spring of 1977 to accomplish these tasks.

B. GOALS AND OBJECTIVES

The primary goal of this thesis project was implementation of an operating system on the Sycor 440 microcomputer system capable of supporting program development at the NPS microcomputer laboratory. Realization of this goal required the debugging, modification, and amplification of the basic MTS program. Since the amount of debugging effort required to achieve the primary objective of the thesis could not be accurately determined, several subgoals were established which would be undertaken if adequate time was available. These subgoals included the integration of the serial printer, the cassette

tape drive and the RS-232 compatible asynchronous communication interface with the basic MTS program.

C. PROBLEM DEFINITION

Since an undetermined proportion of the work required to accomplish the primary goal of this thesis would be the debugging of a major software project, it was decided to improve upon the debugging tools available. At the outset, the only tool available was the Sycor Model 340 Communications Terminal. This terminal was designed for an earlier version of the Sycor 440 system and the scope of its useful debugging commands was quite limited. Therefore, the development of the MTS Debugger Tool (MDT) was undertaken. Written in a systems development language called ML-80, written by L.R.B. Pedroso at NPS in 1975, this tool greatly assisted in the accomplishment of the thesis goals. A detailed description of ML-80 is contained in Ref. 11.

Once debugging of the basic MTS program was accomplished, several test programs were written to check the various MTS service calls; in particular, the disk and terminal I/O functions. Flawless operation of these routines was required prior to implementing an operating system with MTS. A detailed description of the steps required for the implementation of a user operating system on the Sycor 440 Clustered Terminal System is contained in the later chapters.

II. SYCOR 440 HARDWARE DESCRIPTION

The Sycor 440 Clustered Terminal Processing System located in the NPS microcomputer laboratory consists of a Centronix serial printer, four display terminals, the Sycor model 340 communications terminal, an RS-232 asynchronous communications interface, and a desk-high control unit containing a cassette drive. The 440 control unit contains the system logic and memory, consisting of two Intel 8080 processor chips, 64K of random access memory (RAM), driver interfaces for all peripherals, a five megabyte mini-disk and the cassette tape drive.

One of the two 8080's located in the 440 control unit serves as a controller for the five megabyte mini-disk. The mini-disk, which is the primary auxiliary storage device for the 440 system holds the system software including both the Sycor and the MTS operating systems as well as the users' files. The mini-disk is a single platter, movable head disk which is segmented in 512 byte sectors. There are 800 tracks on the disk with 13 sectors per track. Data transfer between RAM and the mini-disk is via direct memory access (DMA). The mini-disk controller communicates with the host 8080 CPU through a 13 byte disk control block (DCB) located at a fixed location in memory [1].

The other 8080 chip found in the 440 control unit serves as the system CPU. The 8080 intruction set consists of 78

data transfer, arithmetic, logical, branch, stack, I/O, and machine control instructions [8]. A comprehensive set of interrupts including timer, auxiliary storage, and peripheral device interrupts are provided by the Sycor 440. Control information and data are exchanged between the 8080 CPU and the Sycor peripheral devices through the I/O ports, or latches, provided on the 8080 chip.

The Sycor 440 system supports synchronous and asynchronous communication devices, up to eight display terminals, serial and line printers and card readers. Presently, the Sycor 440 system at NPS is configured with four 4412 display terminals consisting of a keyboard and CRT display screen. Each terminal is capable of displaying an eight line image of a 576 byte terminal buffer which is located in RAM. Also included is a Centronix serial printer which allows hard copy output of files under both the Sycor and the MTS operating systems. In addition to the mini-disk, auxiliary storage on the Sycor 440 is provided by a cassette drive which is located in the control unit. This drive provides a means for loading the Sycor system software onto the mini-disk. The current Sycor 440 systems configuration also includes an RS-232 communications interface which can, via telephone line, provide a data link to both the PDP-11 minicomputer and the school's IBM 360. Finally, the Sycor 440 is equipped with a model 340 communications terminal. This device can be utilized as a hardware debugger and it also provides a software package which includes provisions for loading and dumping hex format

program files between cassette tape and 440 RAM and for translating hex format tapes into the Sycor relocatable format. This last function of the 340 terminal is important since a mini-disk file must be in the relocatable format to be properly loaded into RAM by the Sycor Loader.

The Sycor 440 system can support other peripherals including magnetic tape drives, floppy disk drives, and card readers; however, these devices are not presently included in the Sycor hardware in the NPS microcomputer laboratory.

III. MTS DESCRIPTION

A. BACKGROUND

MTS was developed in order to integrate the Sycor 440 system into the tutorial and development activities at NPS. Whenever possible, MTS utilizes the Sycor file system to avoid the duplication of system facilities. MTS provides a man-machine interface at the display terminals which is simple, flexible, and convenient, incorporating the best features of interactive computer systems at NPS.

B. MTS INTERNAL DESIGN FEATURES

MTS was designed to provide a timeshared, virtual 8080 microcomputer environment for microcomputer system development. As a timesharing system, MTS is characterized by the following features:

(1) the use of swapping to implement multiprogramming

(2) the use of interrupt driven processor management based on a round-robin scheduling algorithm

(3) the use of virtual floppy disks as the primary auxiliary storage medium

(4) the sharing of a single dedicated I/O device by multiple users [1].

A brief description of each of these features is given in the remainder of this chapter.

Swapping was chosen as the memory management technique employed by MTS. Since the Sycor 440 does not provide address translation hardware, it was not possible to implement the techniques of either paging or dynamic partitioning of memory. Static partitioning of memory was considered, but since the Sycor 440 provides no memory protection in the form of bounds registers, one task executing in memory could access another without detection. Swapping provides physical as well as logical seperation of user tasks. Each of up to four tasks has a mini-disk file associated with it. At any time a task image may reside in RAM or in its swap file on the mini-disk, but at no time can two task images reside in memory simultaneously; thus, task integrity is maintained. An undesirable effect of this design is that the mini-disk transfer rate limits system responsiveness. An improved mini-disk controller is currently being developed by Sycor Inc. Initial tests have shown that this controller will improve the mini-disk transfer rate by a factor of from three to four.

In order to guarantee equitable allocation of the CPU resource to all active tasks, process management of MTS was designed around an interrupt driven task scheduler. A task retains control of the CPU until a hardware timer generates an interrupt, signaling the end of the task's timeslice. The timer interrupt handler then transfers control to the task scheduler to select a new task for execution. To protect the active task, a software lock is set to prevent swapping until the active task returns from an MTS service

call at which time swapping is enabled.

Virtual floppy disks provide auxiliary storage for user programs. These simulated disks consist of 77 tracks, each containing 26 sectors. Each sector is made up of 128 bytes to give a total capacity of 256K bytes of storage. The size of the virtual disk storage area was chosen to reflect the actual size of a physical floppy disk, but can be changed to any conveneient size. Transfer of information between memory and the virtual floppy disk is accomplished through a DMA buffer in the user's memory space. MTS utilizes a mapping function to convert a sector address into a mini-disk sector number. Each user has up to eight virtual floppy disk drives available for use with the virtual disks.

The last major MTS feature is the sharing of dedicated devices by multiple users. Since the cost of a microcomputer CPU makes up such a small percentage of the overall microcomputer system cost, multiprocessing, rather than multiprogramming, should be the logical choice for the design of a timeshared microcomputer system. Since the implementation of this concept was not possible with the Sycor 440 hardware, MTS emphasizes the sharing of the remaining system resources. The sharing of the Sycor 440 memory has already been discussed and is implemented with the use of swapping, providing up to 48K of RAM for user programs. Sharing of the mini-disk auxiliary storage is provided through the use of the virtual floppy disk concept. MTS also allows for the sharing of the Centronix serial printer, which is discussed in chapter V. A detailed

discussion and description of the original MTS design and its implementation on the Sycor 440 hardware can be found in Ref. 1.

Since MTS was designed to support a user operating system, and since certain modifications and enhancements were necessary prior to incorporating an operating system with MTS, thesis work was continued on the original MTS program. The remaining chapters describe the steps which were taken to integrate an operating system into the Sycor 440 / MTS environment.

IV. SYSTEM DEBUGGING TOOLS

A. SYCOR HARDWARE

At the outset of this thesis project, the task of debugging MTS and incorporating a user operating system seemed to be a well defined task; however, it soon became apparent that the tools available for debugging MTS system code were extremely limited. The Sycor 340 communications terminal provided the ability to halt program execution at a specified memory location or breakpoint, to examine the contents of memory locations and to modify the contents of memory locations prior to resuming program execution. Although these features proved to be extremely valuable, greater debugging capability was necessary to realize the thesis objectives. Efficient debugging required the ability to examine the general purpose registers and the program status word (PSW) which contains the accumulator and the carry, zero, parity, plus, and minus flag bits. To effectively debug conditional logic, the ability to specify more than one breakpoint was required. Other debugging functions which were needed included the capabilities of filling memory, moving memory, and transferring information between mini-disk and RAM.

B. MTS DEBUGGER TOOL (MDT)

For the reasons listed above, the MTS Debugger Tool was developed as a first step in the thesis project. This software tool was written as a distinct system module which interfaces directly with MTS. The program utilizes MTS service calls for display terminal and mini-disk I/O and for other system functions. MDT incorporates the option of setting two breakpoints in memory to check program branching. Another feature of the MDT is the ability to start or to resume execution of a system program with interrupts disabled. This feature was necessitated for debugging sections of MTS code where the system interrupts are disabled. Additional MDT facilities include filling memory, setting the contents of memory locations, displaying contiguous 64 byte images of memory at the terminal, reading a specified floppy disk from the mini-disk into memory, and transferring data between RAM and the mini-disk.

The MDT greatly enhanced the debugging phase necessary for providing an operating system on the Sycor 440 hardware for student use at NPS. The debugger resides in the system area of memory where it is still available should the need for it arise. A detailed guide for the use of the MDT is contained in the comment section of the source program which is included with the MTS Program Listings.

V. MTS ENHANCEMENT

Developing the MTS debugger not only facilitated the debugging of MTS code, but it also hastened the remaining development of the basic MTS program. With the aid of MDT several procedures in the monitor and the service modules of MTS were modified to accommodate an operating system. Major areas of modification to the original MTS code are discussed in the following sections.

A. FILE INTEGRITY

Procedures for the protection, unprotection, and restriction of virtual floppy disks with a four character "passkey" were developed to replace the dummy procedures in the original MTS code. It was felt that the completion of these routines, which provide for read and write protection of user file space, was essential since MTS would eventually support a multiuser environment. A detailed explanation of these MTS commands is presented in the MTS User's Guide.

B. SYSTEM LOAD PROCEDURE

Another enhancement to MTS involved a modification to the MTS / SYCOR interface which permits the loading of MTS from the Sycor 440 mini-disk. This modification which was made to the MTS initial program load routine causes a

relocation of MTS code from 1200H to 0H in memory as the system is loaded. This shift of code is necessary for the successful operation of MTS code. This change resulted in an improved user environment and eliminated a tedious five to ten minute load from cassette tape, replacing it with a much simplified ten to fifteen second process. At the same time, this modification allowed for the disconnection of the Sycor 340 communications terminal which was used for loading MTS from cassette tape. Removing dependency on this major piece of system hardware is considered a significant system enhancement.

C. PRINTER INTEGRATION

As the debugging of MTS code neared completion, a decision was made to incorporate the Centronix serial printer as part of the system. Previously, a microcomputer user who desired a hard copy of a program needed to interface with the PDP-11 minicomputer's line printer. Since there is only one data transmission line between the PDP-11 and the microcomputers located in the NPS microcomputer laboratory, only the user working at the microcomputer system with this data line connection could access the PDP-11's line printer. Thus, a user with both microcomputer and and PDP-11 familiarity could, after some inconvenience, obtain a printed listing. Obviously, this user environment is not satifactory for the programmer who desires to work only with a microcomputer system. For this

reason, line printer incorporation with MTS was considered essential.

Certain factors greatly affected the design of the printer interface with MTS. Originally, the concept of using spool files for each user desiring the use of the printer was given much consideration. This concept was in keeping with the overall virtual design of MTS. It would allow a user to output his text to a printer spool file on the mini-disk and then permit him to continue work at his terminal. The information in the spool file would be printed as the printer became available. In theory, the concept of spooling printer information for each user seemed an ideal solution to printer management and was compatible with the overall design philosophy of MTS. However, the operation of the MTS swapping function showed that the mini-disk access times were too high to provide efficient direct memory access (DMA) for large spool files. The fact that the memory buffer for the output to a printer file would be 512 bytes, the length of a mini-disk sector, meant that a considerable number of mini-disk accesses would be required to write to a spool file and to output spool file information to the printer. Several users making simultaneous requests to use the printer would cause a bottleneck in the overall system operation. For this reason, it was decided to dedicate the printer to the first user requesting the device and to advise all other users requesting the printer that the device is in use. Since all four terminals are located in the same area of the

microcomputer laboratory, this decision seemed to be a reasonable alternative.

Because of provisions made in the original design of MTS, implementation of the printer routine was accomplished with few problems. A 512 byte buffer area was available between locations 100H and 300H in the system area of memory. The printer routine was coded such that each time this buffer is filled with data directed for the printer, the contents of the buffer is saved in a specific print file on the Sycor mini-disk. This process is continued until the control-Z character is transmitted indicating an end of file (EOF). The EOF character triggers the initial data output to the line printer. Once the intitial character has been output, the printer interrupt causes the remaining data to be output to the printer on an interrupt basis. When the EOF character is encountered, the printer control flag is turned off and the printer buffer pointers are reset.

The recovery feature of MTS was modified to accommodate the serial printer should a system crash occur during the output of information to the printer. This feature is made possible since all of the printer parameters are saved in the recovery file with the other system parameters needed to recover MTS after a system crash. If the user who has control of the printer at the time of the malfunction is not the task which caused the problem then his printer output will be reinitiated when MTS recovers.

VI. IMPLEMENTATION OF AN OPERATING SYSTEM

A. OPERATING SYSTEM SELECTION

When the debugging of MTS code was near completion the modification of an operating system for the Sycor system which would support program development in the NPS microcomputer laboratory could be undertaken. Initially, a decision was made as to which operating system to implement with MTS, the choices being either ISIS-II, which is the Intel Corp. system or CP/M, which was developed by Digital Research. Refs. 2 and 15 contain information pertaining to these systems. Since the basic MTS design would have to be modified to accommodate ISIS-II and because the CP/M operating system with its support programs was already in use in the NPS microcomputer laboratory, a decision was made to incorporate CP/M on the Sycor 440 / MTS system.

B. OPERATING SYSTEM IMPLEMENTATION

Prior to loading the CP/M operating system onto the Sycor 440 mini-disk certain modifications were necessary to account for the difference in system addresses between the Intellec 8 type microcomputer where CP/M was currently being used and the Sycor 440 system with resident MTS. Following the guidelines of Refs. 4 and 5, a customized basic I/O program (CBIOS), a customized CP/M loader program (CLOAD),

and a program to store CBIOS and CLOAD onto the mini-disk (PUISYS) were written with address and offset changes to match CP/M with the Sycor system.

The main difference between the two systems is that the version of CP/M which was written for the Intellec 8 Mod 80 microcomputer is stored starting at memory location 0 with the command control program (CCP) and the basic disk operating system (BDOS) starting at hexadecimal location 2900H. Also, the user or transient program area (TPA) begins at hexadecimal location 100H. On the Sycor 440 CP/M has been modified such that code is stored at memory location 4000H with CCP and BDOS stored starting at location 6900H in memory. The TPA is based at memory location 4100H. This change allows for the loading of MTS code between 0H and 4000H. An exception to this description of memory storage occurs if a user desires a larger memory image of 32K bytes or 48K bytes of RAM. In these cases, the starting location of the CCP and BDOS modules would be at either A900H or F900H, depending on the RAM image size desired. The CBIOS and CLOAD programs for these larger CP/M systems must also be modified to accommodate these changes. A depiction of a memory image of CP/M for a 16K system with MTS loaded is shown below.

```
         *******************************
         *           CBIOS             *
7E00H    *-----------------------------*
         *           BDOS              *
7200H    *-----------------------------*
         *            CCP              *
6900H    *-----------------------------*
         *                             *
         *            TPA              *
         *                             *
4100H    *-----------------------------*
4000H    *-----------------------------*
         *                             *
         *            MTS              *
         *                             *
         *                             *
   0H    *******************************
```

CP/M MEMORY MAP

Once the CP/M operating system was stored on disk, the task of recompiling various utilty programs was started. The text editor (ED) and status (STAT) programs were the first programs to be modified and loaded on the mini-disk. Regeneration of an object module for these utility programs with a load address of 4100H vice 100H was the only code change made to the utility programs. After a new hexadecimal object file was generated and stored in the PDP-11 file space, the Sycor 440 Interactive Teletype Simulator (ITS) program was utilized to transport the object code via telephone line from the PDP-11 to the Sycor mini-disk. The MDT was then used to read the desired utility program from disk into memory. Once the new program was in memory, the save function of CP/M was utilized to write memory onto the virtual disk, creating a file entry in the CP/M file directory.

VII. CONCLUSIONS AND RECOMMENDATIONS

The primary goal of this thesis project was to implement an operating system on the Sycor 440 microcomputer system capable of supporting program development in the NPS microcomputer laboratory. Enroute to this goal, the debugging and modification of existing system programs which had been written during the design and development phase of the Microcomputer Timeshared System was necessary.

The first step toward this goal was the development of the MTS debugging tool. This software debugger complemented the existing Sycor model 340 terminal hardware debugger and facilitated the debugging and expansion of the original MTS code.

The design of MTS provided areas for enhancements to the basic system. Procedures were written to protect, restrict, and remove protection from user file space. It was felt that these routines were essential to provide file security on a multiuser system. The Centronix serial printer was integrated into MTS and is the only printer in the NPS microcomputer laboratory which directly interfaces with a microcomputer system. This additional resource represents a valuable tool for microcomputer program development.

Following the guidelines of Refs. 4 and 5 the modification of CP/M was accomplished and the revised operating system was loaded onto the Sycor 440 system.

During the implementation of CP/M on the Sycor 440, it was found that certain enhancements to the Sycor hardware would greatly improve the overall system performance. These enhancements include a fast multisector mini-disk controller which is presently under development at Sycor Inc. This enhancement should be given high priority since the response time of the system with three or four users logged into MTS / CP/M is noticably slower than most timeshared systems. Also, to assist in future integration of the Sycor 440 with the present microcomputer facilities, the addition of the Sycor floppy disk drive would be quite beneficial.

A secondary goal of the thesis project was to continue an evaluation of ML-80 as a large system development language. During the modification of MTS code, it was found that ML-80 provided many advantages over assembly language for microcomputer programming. The algebraic notation provided by the language proved to be especially convenient for working at the register level. Also the readability of source code was extremely valuable during the initial phase of studying and understanding MTS. The drawbacks of ML-80 listed in chapter VI of Ref. 1 were found to be valid.

Besides the recommendations for improved hardware mentioned above, there are several areas for future MTS development. These areas include the integration of the cassette tape drive and the RS-232 asynchronous communications interface into the existing MTS configuration.

APPENDIX A MTS USER'S MANUAL

The purpose of this document is to provide the user with the information necessary to utilize the Microcomputer Timeshared System (MTS). This manual was originally prepared by K. J. Brown and D. R. Bullock in March 1977 and revised by S. J. Carro and B. L. Knouse in September 1977.

The contents of this manual include information on setting up the Sycor 440 System for use with MTS, loading and initializing MTS, and interfacing with the MTS and CP/M operating systems. Sections A and B provide a general description of MTS design concepts and the Sycor 440 System. Section C provides the detailed information necessary to interface the Sycor 440 System and MTS. Section D contains information on the terminal design, key functions, and system commands which enable the terminal user to communicate with MTS. Section E describes the MTS status line display and defines the various messages displayed by MTS. Section F details the services provided for a user program by MTS, and the limitations on a user program running in the MTS environment. Section G provides the information required to convert programs written for the basic version of CP/M to the MTS version of CP/M. The complete MTS design specification and implementation information is contained in Ref. 1 and Ref. 2.

TABLE OF CONTENTS

A. MTS CONCEPTS AND DEFINITIONS.................... 32

B. SYCOR 440 HARDWARE DESCRIPTION.................. 33

C. SYCOR 440/MTS INTERFACE......................... 37

1. Loading the System........................... 37

2. Recovery File - .MTSRCVR..................... 38

3. Swap Files - .MTSSWPx........................ 39

4. Configuration File - .MTSCNFG................ 41

5. Printer File - .MTSPRT....................... 44

6. Virtual Floppy Disk Files.................... 44

D. MTS/USER TERMINAL INTERFACE..................... 46

1. Terminal Interface Design.................... 46

a. Status Line.............................. 46

b. Display Buffer........................... 47

c. Terminal Alerts.......................... 48

2. Terminal Key Functions....................... 49

a. Character String Keys.................... 50

b. Entry Mode Keys.......................... 51

c. Line Termination Keys.................... 52

d. Line Editing and Cursor Control Keys... 52

e. Number Pad Keys.......................... 53

3. MTS System Commands.......................... 53

a. General Characteristics.................. 54

b. Syntax Rules............................. 54

c. Parameter Definitions.................... 55

d. Default Parameter Values................. 56

e. Command Descriptions..................... 57

E. MTS STATUS LINE MESSAGES........................ 65
1. Virtual Floppy Disk Status Display......... 65
2. Memory Size Display......................... 66
3. Error Message Display....................... 66
F. MTS/USER PROGRAM INTERFACE...................... 69
1. Program Interface Design.................... 69
2. System Calls................................ 71
a. Arguments................................ 71
b. System Call Descriptions................. 72
3. Service Calls............................... 81
a. Virtual Terminals........................ 81
b. Virtual Floppy Disk Drives............... 83
c. Dedicated Printer........................ 84
d. Arguments................................ 85
e. Service Call Descriptions................ 85
4. Calling Procedure........................... 96
a. 8080 Assembly Language................... 96
b. ML80..................................... 97
c. PL/M..................................... 98
5. Limitations on User Programs................100
G. CP/M - MTS CP/M INTERFACE.......................101
1. CP/M to MTS CP/M Program Conversion........101
2. PRT Program.................................101
H. REFERENCES......................................103

A. MTS CONCEPTS AND DEFINITIONS

The acquisition of the Sycor 440 Clustered Terminal Processing System provided an opportunity for development of a shared environment for microcomputer research and development. In response, the Microcomputer Timeshared System (MTS) was designed and built to provide the basic machine interface and system management functions necessary for a shared environment.

The purpose of MTS is to provide an interface between the bare Sycor 440 machine and up to four user tasks executing concurrently. The MTS environment, as viewed by the user, provides all the microprocessor facilities required for microcomputer research and development. From a system point of view, MTS manages the available hardware to ensure that the hardware resources are equitably and efficiently allocated to competing user programs. MTS is designed to interface with a version of CP/M modified to run on the Sycor 440. This enables all systems and programs designed to run with the CP/M operating system to run on the Sycor 440 with minor modifications (such as a change in load address). This includes all the development facilities available with CP/M, such as the text editor, dynamic debugger, assembler, etc. A list of references for CP/M and its facilities is contained in section G Ref. 3 through Ref. 8.

B. SYCOR 440 HARDWARE DESCRIPTION

The Sycor 440 Clustered Terminal Processing System at NPS is composed of a control unit containing a cassette tape drive, four display terminals, a Centronix matrix printer, and a Sycor Model 340 Communications Terminal.

The control unit is the heart of the 440 system. Contained within a waist-high cabinet are random and control logic including two 8080 chips, 64K of random access memory (RAM), interfaces for all peripheral devices, a five megabyte fixed disk, as well as the aforementioned cassette tape drive.

Located together on the front of the control unit are an ON/OFF/RESET keylock and system status panel. Turning the keylock to the RESET position activates a diagnostic bootstrap program located in read-only memory (ROM). This bootstrap program performs several diagnostic tests on the CPU, memory, and system load device (cassette or mini-disk) and then initiates system loading. The status of the diagnostic tests is indicated by a series of red lights on the system status panel. These lights are turned off in sequence as each phase of the test is successfully completed. When all red lights have been turned off, three green lights on the panel will remain lit to indicate that all power supplies are functioning normally. There is also one additional red light at the bottom of the system status panel which only comes on if the temperature inside the control unit cabinet exceeds normal operating limits.

One of the two 8080 chips located in the 440 control unit serves as the system CPU. The 8080 instruction set consists of the 78 data transfer, arithmetic, logical, branch, stack, I/O, and machine control instructions described in Ref. 9. The Sycor 440 provides a comprehensive set of prioritized interrupts including a timer, peripheral device, and auxiliary storage device interrupts. Passing control information and data between the 8080 CPU and peripheral devices is accomplished by utilizing the I/O ports (called latches in Sycor literature) provided on the 8080 chip.

The second 8080 chip found in the control unit acts as a controller for the mini-disk. The mini-disk is a single platter, movable head, fixed disk blocked into 512 byte sectors. There are 800 tracks on the disk with 13 sectors per track. Data transfer between RAM and the mini-disk is via direct memory access (DMA). The mini-disk controller communicates with the host 8080 CPU through a 13 byte disk control block (DCB) located at a fixed location in memory.

Peripherals supported by the Sycor 440 system include binary synchronous and asynchronous communication devices, up to eight display terminals, serial and line printers, and card readers. The NPS configuration has four display terminals consisting of a typewriter-like keyboard and CRT display device. Each terminal displays a DMA image of a 576 byte terminal buffer located in RAM. Keyboard input is accomplished by software translation of a keyboard matrix code into the corresponding ASCII character code. For

hardcopy output the NPS 440 includes a Centronix serial matrix printer.

Several different auxiliary storage devices may be attached to the Sycor 440 in addition to the mini-disk. These include magnetic tape drives, cassette tape drives, and floppy disk drives. The NPS configuration includes a cassette tape drive located in the control unit. This drive provides compatibility between the Sycor 440 system and the Model 340 debugger.

The Model 340 Communications Terminal is a complete system in its own right which is marketed by Sycor for remote job entry (RJE) applications [10]. When utilized as a hardware debugger, the 340 is augmented with 4K of RAM and a backplane coupling to a special wire-wrapped interface board in the 440 control unit. The 340 debugger is provided with a software package which includes provisions for loading and dumping hex format program files between cassette tape and 440 RAM, examination and modification of individual locations in 440 memory, inserting breakpoints and traps in programs executing on the 440, and single-stepping through a program executing one instruction at a time [12].

There are several hardware characteristics of the Sycor 440 system which strongly influenced the implementation of MTS. The most important of these are:

(1) 8080 CPU architecture

(2) terminal design

(3) mini-disk interface

(4) single-state CPU

(5) lack of memory protection

The impact which each characteristic had on the design and implementation of MTS is covered in Ref. 1.

C. SYCOR 440/MTS INTERFACE

The Microcomputer Timeshared System was designed and built for use on the Sycor 440 Clustered Terminal Processing System. For this reason MTS depends heavily on specific features of the Sycor implementation of an 8080 based microprocessor. This dependence includes reliance on Sycor supplied software as well as the 440 hardware, but becomes most apparent to the user in the two areas of loading the system and maintaining system files.

1. Loading the System

The MTS object module resides on the mini-disk in a relocatable format acceptable to the Sycor System Loader. The System Loader is called in to memory by setting the internal system definition switch to 3 and turning the ON/OFF/RESET keylock on the control unit to the RESET position. After MTS is loaded execution begins with the initial program load (IPL) module. The query RECOVERY? (Y/N) is displayed at terminal 0. The operator should enter Y if recovery is desired, otherwise N. In the event that the IPL operation is halted due to a file access error (file non-existent or cannot be read) the message IPL ABORTED followed by a system file name will appear at terminal 0. After correcting the problem the operator may reload in the normal manner. When the IPL ABORTED message is accompanied by the HARDWARE ERROR terminal alert it indicates that an abnormal completion code was returned by the mini-disk controller after a read operation. Further investigation

using the Sycor utility programs FIXNAR or ..ZAP may be required to identify the problem [11].

To summarize, loading MTS involves the following steps:

(1) set internal system definition switch to 3

(2) turn ON/OFF/RESET keylock to RESET

(3) wait two minutes while mini-disk reaches operating speed and all red lights on control unit status panel go out

(4) respond to RECOVERY? query with N to initialize a new system or Y to recover from the last operating session.

Loading the Sycor operating system involves the following steps:

(1) set internal definition switch to 1

(2) turn ON/OFF/RESET keylock to RESET

(3) wait two minutes while mini-disk reaches operating speed and all red lights on the control unit status panel go out

(4) the Sycor operating system will respond with the prompt READY.

2. Recovery File - .MTSRCVP

MTS supports limited recovery after a user task causes a system crash. The recovery feature is implemented by copying the contents of the system state block (SSB) after each swap to a mini-disk file known as the recovery file. Since the SSB defines the state of the system at any

instant, recovery may be accomplished by reloading the SSB from the recovery file, deleting the task causing the crash, and proceeding with normal execution. These actions are performed by the MTS IPL module when the answer to the RECOVERY? query is Y.

Whenever MTS is running, a double-sector file named .MTSRCVR must be listed in the mini-disk directory. In the event this file is deleted, it may be recreated under the Sycor operating system by using the command

CREATE .MTSRCVR N=2

The contents of the recovery file at the completion of an operating session are only meaningful if recovery will be requested when MTS is next loaded. Therefore, under normal circumstances this file is not needed when MTS is not running.

3. Swap Files - .MTSSWPx

One of the most fundamental requirements on any timesharing system is maintaining independence of user tasks executing concurrently. MTS satisfies this requirement by maintaining physical as well as logical separation of all user tasks in the system. Associated with each of the four terminal tasks is a mini-disk file used to store a core-image of the task when it is waiting for the CPU or blocked pending some I/O operation. At any given instant a task may reside on the disk in its swap file or in memory, but at no time can two or more tasks reside in memory simultaneously.

A task's swap image consists of 17 bytes reserved by

MTS for environment and virtual device control data followed by up to 48,896 bytes of user task memory image. Thus, the maximum allowable swap image is approximately 48K bytes. There is no minimum value for the size of a swap image. The swap image size or, equivalently, the amount of memory space available to the user is variable from 0 to 48K. In fact, the user is encouraged to use the smallest swap image which satisfies his requirements as smaller swap images tend to improve system responsiveness.

A 48K swap image will fill 96 sectors on the mini-disk. Therefore each of the four swap files should normally be 96 sectors long. In the event that mini-disk space is limited, or that user tasks do not requre a 48K swap image, MTS will automatically adjust to any file size greater than 16K (32 sectors). Sixteen kilobytes was selected as the minimum and default system size since it provides a reasonable amount of memory for running the CP/M operating system. The IPL module performs a size test on each swap file to ensure that it is at least 32 sectors long. MTS cannot be loaded if any swap file is smaller than 32 sectors.

If it becomes necessary to change the size of any or all swap files, the file(s) must first be deleted from the mini-disk directory. This is accomplished under the Sycor operating system using the command

DELETE <filename>

where <filename> may be .MTSSWP0, .MTSSWP1, .MTSSWP2, or .MTSSWP3. The number in each case indicates the terminal

with which the file is associated. After the file has been deleted, it may be recreated by using the command

CREATE <filename> N=96

for each file which has been deleted. If swap files smaller than 48K are desired, the value of N in the CREATE command string should be two times the required memory space in kilobytes, but no less than 32.

The contents of the swap files upon completion of an MTS operating session are only meaningful if recovery will be requested when MTS is next loaded. Under normal circumstances these four files are not required when MTS is not running.

4. Configuration File - .MTSCNFG

As explained in section A.2.c, MTS identifies virtual floppy disks by a logical disk number ranging from 0 to 31. Since each virtual floppy disk actually resides in a mini-disk file created under the Sycor operating system, there must be some mechanism for mapping a logical disk number into a filename contained in the mini-disk directory. This function is performed by the configuration file .MTSCNFG.

The configuration file is made up of 32 entries of thirteen bytes each contained on a single mini-disk sector. Each entry has the format

```
----------------------------------------
|   F I L E N A M E   |  K E Y  | P A |
----------------------------------------
 0                    7 8      11   12
```

where FILENAME is the 0 to 8 byte name of the virtual floppy disk file as it appears in the mini-disk directory; KEY is a 0 to 4 byte protection key; PA is the protection attribute of the virtual disk, i.e. 'P' for read/write protection, 'R' for write protection only (restricted access), and blank for no protection. The logical disk number for each entry is simply the position of that entry within the file. For example, the first entry is assigned logical disk number 0, the last entry 31, and the entry which is preceded by 17 other entries becomes number 17.

The configuration file is read by MTS during the initialization process performed by the IPL module. The filename is extracted from each entry and input to a routine which searches the mini-disk directory. When a match occurs the mini-disk address for the file is read from the directory and entered in a virtual floppy disk map table. If no match occurs for a given configuration file entry, the corresponding logical disk number is marked not available. Any subsequent attempt to access that virtual disk will result in the terminal alert DISK NOT AVAIL (E.3).

Since the information contained in the configuration file is of a permanent nature and can only be recreated with great difficulty, the file .MTSCNFG should never be deleted from the mini-disk file directory. In the event the file is deleted erroneously, a new file may be created under the

Sycor operating system using the CREATE and BATCH commands and a backup cassette labeled ".MTSCNFG". The commands are entered as

CREATE .MTSCNFG N=1

RUN BATCH 1=/CSST 3=.MTSCNFG

with .MTSCNFG mounted in the cassette drive. These commands will build a new file directory entry for .MTSCNFG and establish a basic configuration file with 32 entries of the form .DISKx, where x ranges from 0 to 31.

The configuration file may be modified by two methods.

First, immediately after the basic configuration file has been created and loaded it may be modified when running under the Sycor operating system. Sycor provides a data entry free form mode which allows the terminal operator to examine and modify the contents of the file [11]. Extreme care must be exercised when updating .MTSCNFG to align each entry properly in the file. MTS assumes the file will be in the proper format when read, and makes no attempt to validate individual entries. Note that this method may only be used after the configuration file has been loaded and prior to running MTS. Once MTS has been executed the next method must be used.

The configuration file may be modified utilizing the MTS commands Protect, Restrict, and Unprotect. These commands form the most common method for modification of virtual disk protection attributes. Protect, Restrict, and Unprotect are detailed in section D.3.

5. Printer File - .MTSPRT

MTS supports the use of the Centronix serial matrix printer. The printer feature is implemented by buffering characters in memory until the 512 byte printer buffer is full. Next the buffer is written to the mini-disk file .MTSPRT and the buffering operation is restarted. When the end of file character is received, any data in the buffer is copied to .MTSPRT. Finally, the file is read into memory and output to the printer under interupt control.

In the event that .MTSPRT is deleted, it may be recreated under the Sycor operating system by using the command

CREATE .MTSPRT N=<file size>

where <file size> is the length of the file in 512 byte mini-disk sectors. Because all information is written to the printer file before it is printed, the user must insure that the file size is large enough to handle the worst case situation. If the user discovers the printer file size must be increased, the Sycor operating system commands

DELETE .MTSPRT

CREATE .MTSPRT N=<file size>

may be used.

6. Virtual Floppy Disk Files

Each virtual floppy disk resides on a block of logically contiguous mini-disk sectors. This block must be allocated using the facilities provided by the Sycor operating system, specifically the command

CREATE <filename> N=<file size>

where <filename> is a 1 to 8 character name to be entered in the mini-disk directory and configuration file, and <file size> is the length of the file in 512 byte mini-disk sectors. For the basic configuration file described above the <filename> is of the form .diskx where x ranges from 0 to 31. For the standard 256K byte floppy disk <file size> equals 512, i.e. (256 * 1024)/512=512.

Where a physical floppy disk has a fixed capacity of 256K bytes, an MTS virtual disk may have any convenient size up to 256K bytes. MTS assumes that the disk image is made up of contiguous 128 byte floppy disk sectors starting with track 0 sector 1, proceeding through the 26 sectors of track 0 to track 1 sector 1, and so on until track 76 sector 26 or until the virtual disk file is full. If the virtual disk file size is less than 512 mini-disk sectors, less than 77 floppy disk tracks will be addressable.

It is important to note that MTS only recognizes virtual floppy disk files which are entered in the configuration file. The logical disk number associated with a given virtual floppy disk file is determined by that files position in .MTSCNFG. When the file is initially entered in the configuration file a protection key and protection attribute may also be entered.

D. MTS/USER TERMINAL INTERFACE

1. Terminal Interface Design

The general format of each terminal display is as follows:

```
---------------------------------------------
|0              STATUS LINE              63|
---------------------------------------------
|///////////////////////////////////////////|
---------------------------------------------
|0      D                                63|
---------------------------------------------
|64         I                           127|
---------------------------------------------
|128    B       S                       191|
---------------------------------------------
|192        U       P                   255|
---------------------------------------------
|256            F       L               319|
---------------------------------------------
|320               F        A           383|
---------------------------------------------
|384                   E        Y       447|
---------------------------------------------
|448                        R           511|
---------------------------------------------
```

The numbers are decimal and specify character positions within the status line and display buffer.

a. Status Line

The terminal status line is used by MTS to display three types of status information:

(1) The current virtual drive and floppy disk assignments for that terminal.

(2) The size of the user's swap image, i.e. the amount of memory space currently available.

(3) Error message alerts produced by MTS system commands, or resulting from user program calls on the DISPLAY

MSG service routine (see F.2.).

The status line display format and contents are discussed in detail in section E.

b. Display Buffer

The display buffer can hold up to a maximum of 512 characters. The display buffer also acts as an input buffer, holding the input data until the user's program requests it. Due to the unavoidable delays in user program response caused by swapping and aggravated by the relatively low data transfer rate of the mini-disk, the MTS terminal interface provides character echoing and simple line editing features. This ensures reasonable response times to key activation by the user. Thus, input data can not be considered available to the user's program until an input line termination character has been received by MTS. To establish an input buffer for a program, the user enters the data and terminates the line by hitting the NEWLINE or ENTER keys on the keyboard. This establishes that line as an input buffer available for processing by the user's program. When NEWLINE is entered MTS issues a carriage return and line feed to the terminal screen. ENTER causes termination of the current line but MTS does not output a carriage return and line feed to the terminal screen. ENTER is most effective when CP/M commands are entered because more information can be presented on the MTS terminal. Note that the key combinations 'I/O CTL M' or 'SHIFT CR' (on the number pad) will also result in the termination of an input

line. Either of these keys, as well as NEWLINE and ENTER, may be used for line termination.

Once an input buffer has been established the user may continue to input data on the next line. The user may use any of the line editing or other cursor control features on this new line of input data. However, this new line may not be terminated until the user's program has processed the previous input buffer (see terminal alerts below).

Each character output from the user's program is displayed at the current cursor position. Each output results in all input buffer pointers being reset to the character position at the end of the output data. Thus, new I/O will start at this point. This implies that if the user had been in the middle of entering data when the output occurred, it must be reentered.

c. Terminal Alerts

The MTS terminal interface provides the user with either a visual or audio response to each key depression. Normal visual response is provided by display of the entered character and/or movement of the display cursor. The display cursor is a blinking underscore character which marks the current position on the screen. Data is always entered and displayed at the current cursor position.

The audio responses consist of either a beep or click at the terminal. A terminal beep alert will be

generated for any of the following conditions:

(1) An inout buffer is waiting to be processed by the user's program and the terminal user attempts to terminate a new input line.

(2) An attempt is made to move the cursor back past the start of the current line. For example, attempting to delete the previous line or character after the line has been entered by a termination key will result in a beep.

The terminal click alert is associated with the display scrolling feature. Since the display buffer also acts as an input buffer, scrolling the display when the 512 byte display buffer is full could destroy input data which has not yet been processed. For example, the user could be entering a 512 character string. Upon termination of that input line, MTS will lock out scrolling until the user's program has processed the first 64 characters. This ensures that the input data is not destroyed by the scrolling operation. This scrolling lockout is indicated to the user by a terminal click alert.

2. Terminal Key Functions

The terminal keys fall into five basic functional groups: keys for entry of normal character strings; keys which affect the interpretation of the input key character; input line termination keys; line editing and cursor control keys; and number pad keys. These keys and their functions are described in the following subsections. Within the

function descriptions, "current position" refers to the current cursor position on the screen. Any reference to the display of a character in the current position, also implies that the current position is incremented by one.

a. Character String Keys

KEY/SWITCH	FUNCTION
0-9	Displays the input numeric character at the current position on the screen.
Special Characters	Displays the input special character at the current cursor position on the screen.
A-Z	Displays the input alphabetic characters at the current position on the screen. Alphabetic characters are displayed in upper or lower case depending on the key mode (see SHIFT and FS C under Entry Mode Keys).
Tab/Skip	Displays a (horizonal) tab symbol at the current position on the screen.

b. Entry Mode Keys

KEY/SWITCH	FUNCTION
FUNCTION SELECT(FS)	Defines the interpretation of keys for special functions as defined in this section.
I/O CONTROL (I/O CTL)	Defines the interpretation of keys for special functions as defined in this section. Also used in conjunction with the alphabetic keys to generate appropriate ASCII control codes.
SHIFT	Sets the keyboard to upper or lower case.
FS C	Sets or clears the alphabetic key entry mode to upper or lower case. Functions as a shift key lock.
SHIFT RS (RESTART)	Causes a RST 7 instruction to be executed which transfers control to memory location 4038H. This key is used with the CP/M DDT program.

c. Line Termination Keys

KEY/SWITCH	FUNCTION
NEW LINE; ENTER; I/O CTL M; SHIFT CR;	Terminates the current line and establishes the just completed input line as an input buffer available for processing by the user's program. The cursor is displayed at the left most position of the next line except for ENTER which leaves the cursor at the current position.
ERROR RESET (MTS CMD)	Specifies that the input line which it terminates is to be processed by MTS as a system command.

d. Line Editing and Cursor Control Keys

KEY/SWITCH	FUNCTION
NEXT FMAT; I/O CTL U (Line Delete)	Deletes all characters from the current position back to the start of the current line.
BACK SPACE (Char Delete)	Deletes the previously entered character.
FS $ or I/OCTL $	Clears the display buffer (not

(Clear Screen)	the status line) and leaves the current position at the upper left position of the display buffer.
<--- (Cursor Left)	Moves the current position one to the left. Does not delete previous entry, but allows reentry.
--->(Cursor Right)	Moves the current position one to the right. Does not delete previous entry, but allows reentry.

e. Number Pad Keys

The number pad keys consist of 10 numeric digits and 8 ASCII control characters located on the right side of the keyboard. The digits function in the same manner as the other numeric digits on the keyboard. The ASCII control characters are displayed when the SHIFT key is depressed in conjunction with the appropriate key. The only control characters which affect the display are SHIFT CR and SHIFT RS (see Line Termination Keys).

3. MTS System Commands

System commands are a set of commands which provide the user with a means of communicating with MTS from the terminal. These commands allow the user to login to MTS; quit MTS; attach virtual floppy disks; protect, restrict,

and unprotect virtual floppy disks; and specify the virtual memory size to be used.

a. General Characteristics

A MTS command sequence may be entered anytime after the initialization or reinitialization of MTS. The user enters the desired command sequence, followed by the ERROR RESET key. This signals the operating system that there is an MTS command to be processed. Any errors detected in the command sequence will result in an error alert message displayed in the MTS message field on the status line. Section E describes the MTS status line display and provides a summary of the error messages.

b. Syntax Rules

The following rules should be used to interpret the syntax for each system command given in section D.3.e.

(1) The command may be entered in upper or lower case. MTS converts the commands to upper case for processing.

(2) Each entry in the command sequence must be separated by one or more spaces.

(3) The entire command name may be used to specify the command. However only the first letter of the command is required, as indicated by the underscore in the syntax. MTS validates only the first letter of the command name.

(4) Parameters are shown in lower case and enclosed by inequality signs (< >). Each parameter name is a

variable which must be replaced by the appropriate character string or decimal number entered by the user.

(5) Parameters may be required or optional, depending on the command. Optional parameters are specified by enclosing the parameter in brackets ([ ]). If a parameter is designated as optional, it may be omitted from the command sequence (see section D.3.d).

(6) The designation [<disk nr> [/<key>]] indicates that the entire parameter sequence is optional, and that <disk nr> may appear without /<key>. The converse, however, is not true.

(7) If parameters are entered in a command sequence they must be in the order specified in the syntax. For example, <disk nr> may not be entered before <drive ltr>.

(8) The notation (error reset) at the end of each command string is a reminder to the user that each MTS command sequence must be terminated by the ERROR RESET key.

c. Parameter Definitions

The system commands have four types of parameters:

(1) <drive ltr> - must be one of the alphabetic characters A through H. It specifies one of the eight virtual disk drives available to a terminal user.

(2) <disk nr> - must be a decimal number in the range 0-31. It specifies one of up to 32 virtual floppy

disk files on the Sycor mini-disk.

(3) /<key> - a string of not more than four characters, always preceded by the special character '/' which designates the string as a key parameter. All valid ASCII characters are acceptable including blank, slash (/), and other special characters.

(4) <memory size> - must be a decimal value in the range 0 to 48, which specifies the user's swap image size in kilobytes. The default value for a user swap image is 16K.

d. Default Parameter Values

Certain system commands allow the user to omit the <drive ltr> and/or <disk nr> parameters. In these cases, MTS determines the appropriate drive letter and disk number by scanning its allocation tables for the first available virtual drive or virtual disk, as appropriate. If one is found, it is allocated to the requesting user. Otherwise the appropriate error message is displayed.

The <key> parameter is optional only if the disk requested has no protection attributes specified. Thus there is no default <key> value. As previously mentioned, the default <memory size> parameter is 16K.

e. Command Descriptions

The following pages describe in detail each of the system commands:

(1) ATTACH

(2) LOGIN

(3) PROTECT

(4) QUIT

(5) RESTRICT

(6) SIZE

(7) UNPROTECT

Function:

To attach a virtual floppy disk to a virtual disk drive for use by a user at a specific terminal.

Syntax:

ATTACH [<drive ltr>] [<disk nr> [/<key>]] (error reset)
-

Description:

This system command simulates the physical operation of loading virtual disk <disk nr> into virtual drive <drive ltr>. All parameters are optional, section D.3.d describes the default values when optional parameters are omitted.

Error Messages:

DRIVE NOT AVAIL	-	Drive letter has not been specified and there is no drive presently available for assignment.
DISK NR ERROR	-	Disk number entered is greater than 31.
DISK IN USE	-	Disk number specified is presently allocated with read/write access to another user.
DRIVE LTR ERROR	-	Drive letter entered is not one of the letters A through H.
DISK NOT AVAIL	-	Either disk number has not been specified and there is no disk presently available for assignment; or the specified disk is not available for assignment.
INVALID CMD	-	Syntax error in command sequence.
KEY ERROR	-	The specified disk requires a key and either a key has not been entered or the entered key did not match.

Examples:

ATTACH A 3 /USR1
A C
attach 5 /vd#1
a 1

Function:

Links the terminal user to MTS and provides the initial load of the user's program or operating system (default system is CP/M).

Syntax:

LOGIN [<disk nr> [/<key>]] (error reset)

Description:

This system command notifies MTS that the requesting terminal is now active, and simulates the physical cold-start bootstrap operation of the user's system. The bootstrap load always takes place from virtual drive A. The virtual disk (and associated key, if any) attached to this drive may be specified as a parameter. The default is disk nr 0, which is a read only disk and always contains the CP/M operating system.

Error Messages:

DISK NR ERROR	- Disk number entered is greater than 31.
DISK IN USE	- Disk number specified is presently allocated with read/write access to another user.
DISK NOT AVAIL	- The specified disk is not available for assignment.
HARDWARE ERROR	- Abnormal completion status was returned by the mini-disk controller following a write operation.
INVALID CMD	- Syntax error in command sequence.
KEY ERROR	- The specified disk requires a key and either a key has not been entered of the entered key did not match.

Examples:

LOGIN 3 /01
L 15
login 25 /a25
l

Function:

Adds the read/write protection attribute to the specified virtual disk.

Syntax:

PROTECT <disk nr> /<key> (error reset)
-

Description:

This system command provides the user with the means for on-line assignment of a protection <key> to <disk nr>.

Error Messages:

DISK NR ERROR	- Disk number entered is greater than 31.
HARDWARE ERROR	- Abnormal completion status was returned by the mini-disk controller following a read or write operation.
INVALID CMD	- Syntax error in command sequence.
KEY ERROR	- The specified disk is already protected. To change protection keys use UNPROTECT with current key and then PROTECT with new key.

Examples:

```
PROTECT 1 /VFD5
protect 10 /key1
p 2 /u#20
```

Function:

Terminates the terminal user's link to MTS.

Syntax:

QUIT (error reset)

Description:

This system command notifies MTS that the requesting terminal is no longer active.

Error Messages:

HARDWARE ERROR - Abnormal completion status was returned by the mini-disk controller following a write operation. This may indicate that the last virtual disk written to was not closed properly and data has been lost.

INVALID CMD - Syntax error in command sequence.

Examples:

QUIT
quit
Q
q

Function:

Adds the read restriction attribute to the specified virtual disk.

Syntax:

RESTRICT <disk nr> /<key> (error reset)
-

Description:

This system command provides the user with the means for on-line assignment of a "read only" restriction to <disk nr>. This allows the user to specify a virtual floppy disk as available to other users for read only access. The virtual floppy disk must have been protected prior to issuing the restrict command. The <key> is the same as the <key> used in the protect command.

Error Messages:

DISK NR ERROR - Disk number entered is greated than 31.

DISK IN USE - Disk number specified is presently allocated with read/write access to another user.

INVALID CMD - Syntax error in command sequence.

KEY ERROR - The specified disk requires a key and either a key has not been entered or the entered key did not match.

Examples:

```
RESTRICT 3 /ID 1
R  4  /  ID4
restrict 13 /usr3
r 16 /1
```

Function:

Specifies the memory size to be allocated to the terminal user.

Syntax:

SIZE <memory size> (error reset)

Description:

This system command sets the size of the user's swap image. The range of values is 0-48K. The default value after login is 16K.

Error Messages:

INVALID CMD - Syntax error in command sequence or the size parameter entered does not fall in the range of 0-48; or the Sycor 440 swap file is not large enough to hold this size swap image (see section C.3).

Examples:

```
SIZE 24
S 32
size   48
```

Function:

To remove a previously entered protection key from the specified virtual floppy disk.

Syntax:

UNPROTECT <disk nr> /<key> (error reset)
-

Description:

This system command provides the user with the means for on-line removal of all protection attributes form <disk nr>.

Error Messages:

DISK NR ERROR	- Disk number entered is greater than 31.
HARDWARE ERROR	- Abnormal completion status was returned by the mini-disk controller following a read or write operation.
INVALID CMD	- Syntax error in command sequence.
KEY ERROR	- A protection key is required and either no key was entered, or the entered key did not match.

Examples:

```
UNPROTECT 18 /NR 9
U  12  /0964
unprotect 7 /0vfd
u  9  /0?&%
```

E. MTS STATUS LINE MESSAGES

The MTS operating system utilizes the first line of each terminal for system status and error message displays. The status line is 64 characters in length and is divided into three display areas as shown below.

STATUS LINE

```
 0                   39 40     47 48     63
--------------------------------------------
|        V F D         |   M S   |  M S G  |
--------------------------------------------
```

where

VFD - Virtual Floppy Disk Status Display

MS - Memory Size Display

MSG - Error Message Display

1. Virtual Floppy Disk Status Display

This display contains information on the virtual drive and disk assignments currently in effect. For each attached virtual floppy disk the following will be isplayed:

(1) drive letter

(2) disk number

(3) restriction indicator (r or blank)

For example, if the user has attached disk number 3 to drive A and disk number 25 (which is restricted) to drive C, the status display would appear as follows:

```
 0          10                                39
------------------------------------------- - - -
|A=03       C=25r
------------------------------------------- - - -
```

2. Memory Size Display

The center of the status line display shows the current memory size for that user and the "MTS" header. For example, if the system default memory size were being used, the display would appear as follows:

```
     40     47
- ------------- - - -
     16K MTS
- ------------- - - -
```

3. Error Message Display

All MTS system commands are validated and an error alert generated if any syntax errors are found. The last 16 positions of the status line are reserved for these messages. A valid system command will clear the error message display of any previous error alert. The following is a summary of system error messages.

MESSAGE	MEANING
(Blank Display)	Initial condition; also the status message area is cleared following the processing of a valid system command.
DISK IN USE	Disk number specified is presently allocated with read/write access to another user.

DISK NOT AVAIL	Either the disk number has not been specified and there is no disk presently available for assignment; or specified disk number is not available for assignment.
DISK NR ERROR	Disk number entered is greater than 31.
DRIVE LTR ERROR	Drive number entered is not one of the letters A through H.
DRIVE NOT AVAIL	Drive letter has not been specified and there is no drive presently available for assignment.
HARDWARE ERROR	Abnormal completion status was returned by the mini-disk controller following a read or write operation.
INVALID CMD	Syntax error has been detected in the command sequence.
KEY ERROR	The specified disk requires a key and either a key has not been entered or the entered key did not match.
PRINTER NOT RDY	Abnormal completion status was returned by the printer controller following a print operation. This error may be caused by printer power off, printer not selected or printer out of paper.

TASK DELETED — When RECOVERY is specified during system initialization, this message is displayed at the terminal which was executing when the system failure occurred. It indicates that this terminal user must reestablish the environment.

F. MTS/USER PROGRAM INTERFACE

1. Program Interface Design

MTS was designed to provide a timeshared, virtual 8080 microprocessor environment for microcomputer systems development. The term virtual is appropriate here because the user actually interfaces with MTS for many services normally provided by hardware in a dedicated CPU environment. A software interface between user programs and the Sycor 440 hardware is necessary in order to allocate the hardware resources equitably and efficiently, while at the same time satisfying the service requirements of several competing user programs.

The MTS/user program interface consists of a set of service routines which may be called by a user program through a single entry point to perform terminal I/O, access virtual floppy disks, or modify the user's virtual environment. The design was heavily influenced by the CP/M operating system which uses a similar scheme for I/O.

The set of service routines may be logically divided into two types of calls on MTS. The first type, system calls, perform the same functions for a user program as system commands provide for the user at a terminal (D.3). The functions deal with modifying the user's current virtual environment by changing memory size, attaching various virtual disks to virtual drives, or even logging on and off the system. Service calls are the second type of call provided by the MTS software interface. Service calls are

used to perform terminal I/O and access virtal floppy disks.

A call to MTS takes the form

<value> = MTS(<fid>,<parm>)

The first argument, <fid>, is a number from 0 to 17 which identifies the function requested. The <parm> argument may be a parameter value, if only a single parameter is required, or the address of a parameter list if more than one parameter is required. In each case MTS returns <value> upon completion of the requested operation. This returned value may be an ASCII character code, an error code, or in several cases have no significance. Both system calls and service calls are formed as described above. The syntax and function of each call are described in sections 2 and 3. Section 4 details the calling procedures which include the register assignments for <fid>, <parm> and return values.

2. System Calls

All system commands available to a user at a terminal are also available to a user program through system calls. An additional call is provided which will display an appropriate terminal alert at the user's terminal if entered with an error code. Table 1 summarizes the required arguments and return values of each system call.

TABLE 1

SYSTEM CALL SUMMARY

FID	NAME	PARM	VALUE
0	ATTACH	list	error code
1	DISPLAY MSG	error code	none
2	LOGIN	list	error code
3	PROTECT	list	error code
4	QUIT	none	none
5	RESTRICT	list	error code
6	SIZE	size	error code
7	UNPROTECT	list	error code

a. Arguments

Each system call is identified by a number which MTS associates with a particular service routine. In addition to this function identifier, MTS may require one or several additional parameters to perform the requested service. When more than one parameter is required, MTS must be passed the address of a byte vector containing these parameters. Each system call requires that this vector

conform to some fixed format. In general each byte of the vector will contain some numeric value or an ASCII character code, but situations may arise when an optional parameter is not specified. In such cases the corresponding byte in the parameter vector must be filled with the value FFH. Section 4 details the calling procedure which includes the register assignments for the arguments.

b. System Call Descriptions

The following pages describe in detail each system call.

Function:

To attach a virtual floppy disk to a virtual disk drive for use by a user at a specific terminal.

Arguments:

FID = 0

PARM = address of parameter vector
byte 0: drive number where A=0, B=1, etc.
byte 1: disk number - 0 to 31
bytes 2-5: protection key - 0 to 4 ASCII characters

Description:

This call simulates the physical operation of loading virtual disk <disk nr> into virtual drive <drive nr>. All parameters are optional. If disk and/or drive nr is not specified MTS searches the disk or drive map table respectively for the first available entry. A protection key is only required if the virtual disk to be attached has been assigned read/write protection. The call returns an error code upon completion.

Error Codes:

0 - Operation successful

2 - Either disk number has not been specified and there is no disk presently available for assignment; or the specified disk is not available for assignment.

3 - Disk number specified is presently allocated with read/write access to another user.

4 - Disk number specified is greater than 31.

5 - The specified disk requires a key and either a key has not been entered or the entered key did not match.

6 - Drive number specified is greater than 7.

10 - Drive number has not been specified and there is no drive presently available for assignment.

Function:

To display a terminal alert in the status message area of the user's terminal.

Arguments:

FID = 1

PARM = error code

Description:

This call takes an error code as input and displays the corresponding predefined terminal alert message on the user's terminal. The DISPLAY MSG system call provides the only way for a user to display messages on the terminal status line. No value is returned by this system call.

Action:

0 - blank

1 - INVALID CMD

2 - DISK NOT AVAIL

3 - DISK IN USE

4 - DISK NR ERROR

5 - KEY ERROR

6 - DRIVE LTR ERROR

7 - PRINTER NOT RDY

8 - HARDWARE ERROR

9 - TASK DELETED

10 - DRIVE NOT AVAIL

Function:

Reinitializes the user's MTS environment and reboots the user's system from drive A.

Arguments:

FID = 2

PARM = address of parameter vector
byte 0: disk number - 0 to 31
bytes 1-4: protection key - 0 to 4 ASCII characters

Description:

This system call creates a reinitialized MTS environment for the user program. Memory size remains at the current value and the specified disk, if any, will be attached to drive A. If no disk number is specified, disk number 0 containing the CP/M system is attached. Drive assignment and memory size will be displayed on the status line of the user's terminal. The user system is then rebooted from drive A. All other drives and disks are reset.

Error Codes:

0 - Operation successful

2 - The specified disk is not available for assignment.

3 - Disk number specified is currently allocated with read/write access to another user.

4 - Disk number specified is greater than 31.

5 - The specified disk requires a key and either a key has not been entered or the entered key did not match.

8 - Abnormal completion status was returned by the mini-disk controller following a write operation. This may indicate that the last virtual disk written to was not closed properly and data has been lost.

Function:

Read/write protect a virtual disk.

Arguments:

FID = 3

PARM = address of parameter vector
byte 0: disk number - 0 to 31
bytes 1-4: protection key - 1 to 4 ASCII characters

Description:

This system call adds the read/write protection attribute to the specified virtual disk. This is accomplished by the update of the basic configuration file .MTSCNFG [C.4]. Upon normal completion of the call disk use is limited only to those who know the protection key.

Error Codes:

0 - Operation successful

4 - Disk number specified is greater than 31.

5 - Key has not been entered or disk is already protected with another key.

8 - Abnormal completion status was returned by the mini-disk controller following a read or write operation. This may indicate that the last virtual disk written to was not closed properly and data has been lost.

Function:

Logs the user off MTS.

Arguments:

FID = 4

PARM = none required

Description:

This system call notifies MTS that the requesting user program is no longer active. Control will not be returned to the user program. The user must log in through the terminal to regain system facilities.

Error Codes:

0 - Operation successful.

8 - Abnormal completion status was returned by the mini-disk controller following a write operation. This may indicate that the last virtual disk written to was not closed properly and data has been lost. If this error code is returned, the terminal alert HARDWARE ERROR is automatically displayed in the status message area of the user's terminal.

Function:

Assignment of "read only" restriction to a virtual disk.

Arguments:

FID = 5

PARM = address of parameter vector
byte 0: disk number - 0 to 31
bytes 1-4: protection key - 1 to 4 ASCII characters

Description:

This system call adds the read restriction attribute to a virtual disk which has been previously protected. The protection key is the same as that used for the protect system call. MTS adds the read restriction attribute to the basic configuration file .MTSCNFG.

Error Codes:

0 - Operation successful

3 - Disk number specified currently allocated with read/write access to another user.

4 - Disk number specified is greater than 31.

5 - The specified disk requires a key and either a key has not been entered or the entered key did not match.

8 - Abnormal completion status was returned by the mini-disk controller following a read or write operation. This may indicate that the last virtual disk written to was not closed properly and data has been lost.

Function:

Set the amount of memory available to the user.

Arguments:

FID = 6

PARM = memory size in kilobytes

Description:

This system call adjusts the size of the user's swap image to the specified value. The value must fall in the range 0 to 48, and also must not be greater than the size of the swap file associated with the user's terminal.

Error Codes:

0 - Operation successful.

1 - Either specified size exceeds 48K, or a value less than 48K exceeds the size of the available swap file.

Function:

Remove a previously specified protection key from a virtual disk.

Arguments:

FID = 7

PARM = address of parameter vector
byte 0: disk number - 0 to 31
bytes 1-4: protection key - 1 to 4 ASCII characters

Description:

This system call removes all protection attributes from the specified virtual disk. This is accomplished by modification of the configuration file .MTSCNFG.

Error Codes:

0 - Operation successful

4 - Disk number specified is greater than 31.

5 - A protection key is required and either a key has not been entered or the entered key did not match.

8 - Abnormal completion status was returned by the mini-disk controller following a read or write operation. This may indicate that the last virtual disk written to was not closed properly and data has been lost.

3. Service Calls

The MTS environment currently provides a virtual CRT terminal as the primary I/O device and virtual floppy disk drives for auxiliary storage. Access to both of these virtual devices is through MTS service calls. A summary of the service calls showing parameters and returned values is given in Table 2.

TABLE 2

SERVICE CALL SUMMARY

FID	NAME	PARM	VALUE
8	TERMINAL STATUS	none	true/false
9	READ TERMINAL	none	character
10	WRITE TERMINAL	character	none
11	WRITE PRINTER	character	error code
12	SELECT DRIVE	drive nr	error code
13	SET DMA	dma address	error code
14	SET TRACK	track nr	error code
15	SET SECTOR	sector nr	error code
16	READ FLOPPY	none	error code
17	WRITE FLOPPY	none	error code

a. Virtual Terminals

The MTS virtual terminal simulates the operation of a serial half-duplex console device. Single ASCII characters may be passed from the terminal keyboard to a user program, or from a user program to the terminal for

display. A service call to MTS is required to pass each character. MTS also provides a terminal status service call which allows a user program to test the status of a terminal.

The user should keep in mind that characters are actually being passed between his program and the terminal display buffer (D.1.b). This means that input need not be echoed by the user's program since it already appears on the display. Simple line editing is also provided by MTS on the input data prior to making that data available for processing by the user's program.

The user can directly contribute to improved system response by proper use of the terminal service calls. It is common practice when writing conversational programs to implement a "get character" routine to handle input from the terminal. Normally this routine does little more than repeatedly test the terminal status until it finds input waiting. In the MTS environment a more efficient method of accomplishing the same goal is to immediately read from the terminal without testing for status. If input is waiting, the first character will be passed immediately. More importantly, if there is no input waiting, MTS will block the user's program until a character is entered at the keyboard. The blocked program may be swapped out and the CPU allocated to another user. This method of implementing conversational programs takes advantage of unproductive waiting time in one user program to service additional users.

b. Virtual Floppy Disk Drives

The MTS virtual floppy disk drive provides auxiliary storage for user programs on virtual floppy disks. These hard-sectored disks have 128 bytes per sector, 26 sectors per track, and a maximum number of tracks determined by the size of the file containing the disk image (C.6). Each user has eight drives available for dedicated use.

Drive A is activated when the user logs in and serves as the user system load device. In a process which simulates a cold-start bootstrap load the first four sectors on track 0 are read into the user's memory space at location 4000H. MTS assumes that these sectors contain executable code which will load the remainder of the user's system. Unless another disk is specified in the LOGIN command string, a read-only disk containing the CP/M operating system will be attached when drive A is activated.

The user may activate any or all of the remaining virtual drives by attaching a virtual disk. This is accomplished from the terminal by entering the ATTACH system command or directly from the user's program by a call to MTS. Although no direct method for detaching a virtual disk is provided by MTS, the same effect is produced indirectly by overriding the current drive assignment with a second ATTACH command. When the second floppy disk is attached MTS closes the previously attached disk and releases it for use elsewhere.

Data transfer between a virtual disk and a user program utilizes a 128 byte buffer in the user's program

space called a DMA buffer. The name is derived from the nature of the transfer operation: to the user program it appears that data transfer is by direct memory access.

Before the user program can access a particular virtual disk sector the user must specify a complete sector address and a DMA buffer. A complete sector address consists of drive, track, and sector numbers. Note that MTS will not allow a virtual drive to be selected until a disk has been attached. A DMA buffer is defined by its base address. MTS provides a service call to enter each of these four values. Once a value has been entered it will be used for all subsequent virtual disk accesses until redefined by a second service call.

c. Dedicated Printer

The MTS dedicated printer provides the user with the ability to obtain hard-copy output. Single ASCII characters may be passed from the user program to the printer for display. A service call to MTS is required to pass each character.

The printer is a dedicated device. This means that once a task is allocated the printer, no other tasks may use the printer. A task is allocated the printer merely by the successful completion of a call to the write printer service routine. The write printer service routine buffers the characters in memory and eventually writes the characters to the .MTSPRT disk file. Note that the printer does not actually print the characters until the end of file

character is received. When the user task issues the end of file character (control Z), the printer prints the information contained in .MTSPRT. A repeat character (control R) causes the information in .MTSPRT to be output to the printer. After the control R or Control Z has been issued the printer is not available to any task until after all information in .MTSPRT has been printed.

d. Arguments

Service calls have the same form as other calls to MTS. A numerical function identifier is associated with each call to identify the service desired. The second argument is a single parameter in most cases, although several of the service calls require no second argument. Section 4 details the calling procedures which include the register assignments for the arguments.

e. Service Call Descriptions

The following pages describe in detail each individual service call.

Function:

Interrogate the status of the user's terminal.

Arguments:

FID = 8

PARM = none required

Description:

This service call returns a logical value answering the question "Is terminal input waiting?" TERMINAL STATUS should not be used in those situations where no further processing can be accomplished until terminal input is available. In such a case it is more efficient to use the READ TERMINAL service call to allow processing of other user tasks while waiting.

Value:

00H - all terminal input processed

FFH - terminal input waiting

Function:

Read the next available character from the user's terminal.

Arguments:

FID = 9

PARM = none required

Description:

This service call passes the next available ASCII character from the user's terminal display buffer to the user program. The maximum size input line is 512 characters. Each input line is terminated by a carriage return. It is not necessary for user programs to echo input characters since they are already displayed on the user's terminal before becoming available to the user program. Line editing functions are provided by MTS.

Value:

A single ASCII character - the end of each input line is indicated by the return of a carriage return (ASCII = 0DH).

Function:

To write a character to the user's terminal.

Arguments:

FID = 10

PARM = a single ASCII character

Description:

This service call passes the specified character from the user program to the terminal display buffer for display. Carriage return (ASCII = 0DH) returns the cursor to first position of the current line. Line Feed (ASCII = 0AH) moves the cursor down one line. Each output line will normally be terminated by the CR-LF combination.

Value:

None returned.

Function:

To write a character to the printer.

Arguments:

FID = 11

PARM = a single ASCII character

Description:

This service call passes the specified character from the user program to the printer buffer. The printer does not print any characters until the end of file character (control Z) is received. If an end of file character had been previously received, the repeat character (control R) may be issued resulting in the printing of information up to the last end of file.

Error Codes:

0 - Operation successful

7 - Printer is not ready. Printer may be turned off, not selected or out of paper.

8 - Abnormal completion status was returned by the mini-disk controller following a write operation.

11 - Printer is in use. This code indicates that another task currently has control of the printer or the printer is currently printing a file.

12 - The end of the printer file .MTSPRT has been exceeded. The user must recreate the .MTSPRT file with a larger file size in order to alleviate this error [C.5].

Function:

Selects the virtual floppy disk drive to be used in subsequent floppy disk accesses.

Arguments:

FID = 12

PARM = drive number where A=0, B=1, etc.

Description:

This service call selects one of the eight virtual floppy disk drives available to each user program for use in subsequent floppy disk accesses. Before a drive can be selected, a virtual disk must be attached.

Error Codes:

0 - Operation Successful.

6 - Drive number specified is greater than 7. Selected drive is changed.

8 - Abnormal completion status was returned by the mini-disk controller following a read or write operation.

10 - Drive specified is not in use. Indicates that no virtual disk has been attached to the specified drive. Selected drive is unchanged.

Function:

Sets the base address of the 128 byte DMA buffer to be used in subsequent floppy disk accesses.

Arguments:

FID =13

PARM = base address of DMA buffer

Description:

The DMA buffer required to access a virtual floppy disk must be a contiguous block of 128 bytes located in the user's memory space, i.e. with base address greater than 4000H. Specifying a DMA address greater than or equal to FF00H will have unpredictable results, but can normally be expected to cause a system crash and subsequent deletion of the user's task upon recovery.

Error Codes:

0 - Operation successful.

12 - Address specified is less than the base of user's memory space. Current DMA address remains unchanged.

Function:

Sets the floppy disk track number to be used in subsequent virtual floppy disk accesses.

Arguments:

FID = 14

PARM = track number - 0 to 76

Description:

This service call sets the track number to be used in subsequent floppy disk accesses. Values may range from 0 to 76. The value cannot be validated until it is associated with a virtual floppy disk number; therefore, no validation is performed until a read or write operation is requested.

Error Codes:

0 - Operation successful

12 - Track number specified is greater than 76. Current value of track number remains unchanged.

Function:

Sets the floppy disk sector number to be used in subsequent virtual floppy disk accesses.

Arguments:

FID = 15

PARM = sector number - 1 to 26

Description:

This service call sets the sector number to be used in subsequent virtual floppy disk accesses. Since each floppy disk track contains 26 sectors numbered from 1 to 26, this value cannot be less than 1 nor greater than 26.

Error Codes:

0 - Operation successful.

12 - Sector number specified is less than 1 or greater than 26. Current value of sector number remains unchanged.

Function:

Simulates reading a 128 byte sector from a floppy disk.

Arguments:

FID = 16

PARM = none required

Description:

This service call simulates reading from a floppy disk by mapping the current drive, track, and sector numbers into a mini-disk address; reading the mini-disk sector into a buffer; and moving 128 bytes from the mini-disk buffer into the current DMA buffer in the user's memory space. Errors may occur at two points in the process. If the calculated mini-disk address falls outside the bounds of the virtual disk file attached to the specified virtual drive, it indicates an error in the specified track number. Errors may also occur during mini-disk read and write operations. A user program must consider such hardware errors as irrecoverable since MTS provides insufficient information to determine the cause.

Error Codes:

0 - Operation successful.

8 - Abnormal completion status was returned by the mini-disk controller following a read or write operation.

12 - Calculated mini-disk address out of bounds. Probable error in specified track number.

Function:

Simulates writing a 128 byte sector to a floppy disk.

Arguments:

FID = 17

PARM = none required

Description:

This service call simulates writing to a floppy disk by mapping the current drive, track, and sector numbers into a mini-disk address; reading the mini-disk sector into a buffer; and moving 128 bytes of data from the current DMA buffer in the user's memory space into the mini-disk buffer. Errors may occur at two points in the process. If the calculated mini-disk address falls outside the bounds of the virtual disk file attached to the specified virtual drive, it indicates an error in the specified track number. Errors may also occur during mini-disk read and write operations. The user should interpret such hardware errors as indicating a bad sector on the virtual floppy disk and try repeating the operation with a different floppy disk sector and track combination.

Error Codes:

0 - Operation successful.

2 - Floppy disk is read only. Write is not permitted.

8 - Abnormal completion status was returned by the mini-disk controller following a read or write operation.

12 - Calculated mini-disk address out of bounds. Probable error in specified track number.

4. Calling Procedure

All calls to MTS, whether system calls or service calls, are made through a single entry point at location 2000H. Each call takes two arguments: the function identifier in register C; and a parameter value or address in register pair DE. In those cases where the second argument is only a single byte the contents of the D register are ignored.

Each call to MTS returns a value in the A register. This value may be an error code, an ASCII character code, or zero. The value zero is returned by those routines whose value has no significance such as WRITE TERMINAL.

Note that the register assignments for arguments and returned values conform to the PL/M convention for passing parameters. The following examples illustrate the calling procedure for 8080 Assembly Language, ML80, and PL/M in the MTS environment. Each example illustrates the sequence required to read floppy disk sector 22, track 43 on drive 2 into a DMA buffer at address 4100H.

a. 8080 Assembly Language

When writing in 8080 assembly language MTS is accessed by a direct call to the MTS entry point:

```
MTS     EQU     2000H
        .
        .
        .
        MVI     C,12        ;FID = 12
        MVI     E,2         ;DRIVE NR = 2
        CALL    MTS         ;SELECT DRIVE
        MVI     C,13        ;FID = 13
```

```
        LXI     D,4100H     ;DMA ADDRESS = 4100H
        CALL    MTS         ;SET DMA
        MVI     C,14        ;FID = 14
        MVI     E,43        ;TRACK NR = 43
        CALL    MTS         ;SET TRACK
        MVI     C,15        ;FID = 15
        MVI     E,22        ;SECTOR NR = 22
        CALL    MTS         ;SET SECTOR
        MVI     C,16        ;FID = 16
        CALL    MTS         ;READ FLOPPY
        .
        .
        .
        MVI     C,15        ;FID = 15
        MVI     E,23        ;CHANGE SECTOR NR
        CALL    MTS         ;SET SECTOR
        MVI     C,16        ;FID = 16
        CALL    MTS         ;READ AGAIN
```

b. ML80

The readibility of ML80 source programs may be enhanced by defining an M80 macro for each call to MTS used in the program. The following code segment contains several examples.

```
[MACRO MTS '2000H']
[MACRO SELECT$DRIVE DNR '
    C = 12; E = [DNR]; CALL [MTS]']
[MACRO SET$DMA DMA '
    C = 13; DE = [DMA]; CALL [MTS]']
[MACRO SET$TRACK TNR '
    C = 14; E = [TNR]; CALL [MTS]']
[MACRO SET$SECTOR SNR '
    C = 15; E = [SNR]; CALL [MTS]']
[MACRO READ$FLOPPY '
    C = 16; CALL [MTS]']
    .
    .
    .
/* SPECIFY COMPLETE SECTOR ADDRESS AND DMA BUFFER */
[SELECT$DRIVE '2'];
[SET$DMA '4100H'];
[SET$TRACK '43'];
[SET$SECTOR '22'];
[READ$FLOPPY];
    .
    .
    .
```

```
/* INCREMENT SECTOR NR AND READ AGAIN */
[SET$SECTOR '23'];
[READ$FLOPPY];
```

c. PL/M

```
/*******************************************************/
/*         SAMPLE PL/M PROGRAM SEGMENT                 */
/*******************************************************/

4000H:

USER: PROCEDURE;
DECLARE
    LIT            LITERALLY 'LITERALLY',
    MTS            LIT       '2000H',
    SELECT$DRIVE   LIT       '12',
    SET$DMA        LIT       '13',
    SET$TRACK      LIT       '14',
    SET$SECTOR     LIT       '15',
    READ$FLOPPY    LIT       '16',
    DISPLAY$MSG    LIT       '1' ;
/********************************************************/
/*              MTS INTERFACE PROCEDURES                */
/********************************************************/

MTS1: PROCEDURE (FID,PARM);
    /************************************/
    /* PROVIDES THE MTS INTERFACE FOR   */
    /* FUNCTIONS WHICH DO NOT REQUIRE A */
    /* RETURN VALUE.                    */
    /* INPUT: FID - MTS FUNCTION ID     */
    /*        PARM - PARAMETER OR ADDRESS*/
    /*               OF PARAMETER LIST. */
    /************************************/
    DECLARE FID BYTE, PARM ADDRESS;
    GO TO MTS;
    END MTS1;

MTS2: PROCEDURE (FID,PARM) BYTE;
    /************************************/
    /* PROVIDES THE MTS INTERFACE FOR   */
    /* FUNCTIONS WHICH REQUIRE A VALUE  */
    /* RETURNED. INPUT PARAMETERS ARE   */
    /* THE SAME AS IN MTS1.             */
    /************************************/
    DECLARE FID BYTE, PARM ADDRESS;
    GO TO MTS;
    END MTS2;
    .
    .
    .
```

```
/*************************************************************/
/*  SPECIFY COMPLETE SECTOR ADDRESS AND DMA BUFFER  */
/*************************************************************/

CALL MTS1(SELECT$DRIVE, 2);
CALL MTS1(SET$DMA, 4100H);
CALL MTS1(SET$TRACK, 43);
CALL MTS1(SET$SECTOR, 22);
/*************************************************************/
/* READ FLOPPY RETURNS AN ERROR CODE WHICH WILL      */
/* BE RETURNED TO MTS TO BE DISPLAYED ON THE         */
/* TERMINAL STATUS LINE.                             */
/*************************************************************/
CALL MTS1(DISPLAY$MSG, MTS2(READ$FLOPPY,0));
     .
     .
     .
/* INCREMENT SECTOR NR AND READ AGAIN                */

CALL MTS1(SET$SECTOR, 23);
CALL MTS1(DISPLAY$MSG, MTS2(READ$FLOPPY,0));
     .
     .
     .
END USER;
     .
     .
     .
EOF
```

5. Limitations on User Programs

MTS was designed to provide each user with his own virtual 8080 microprocessor. Unfortunately, the architecture of the 8080 CPU is not amenable to self-virtualization. As a consequence several limitations must be imposed on user programs running in the MTS environment. These limitations are:

(1) The user's memory space extends from address 4000H to FEFF, a total of 48,896 bytes. All user code, data, and buffers must be contained within this area of memory.

(2) All user-defined stacks must be four bytes longer than the maximum size required by the user. The four extra bytes are needed if an interrupt occurs while the user's stack is full. Failure to provide this additional space may result in random execution errors which are not reproducible and extremely difficult to diagnose.

(3) User programs should not read or write directly to I/O ports while running under MTS. Terminal and floppy disk access is provided by MTS service calls. Attempts to interface directly with the Sycor 440 peripherals or auxiliary storage devices may interfere with the operation of MTS and damage other users.

G. CP/M - MTS CP/M INTERFACE

1. CP/M to MTS CP/M Program Conversion

Programs may be easily converted from execution under the basic CP/M system to execution under the MTS CP/M system. In the basic version of CP/M the base address of the system is 00H whereas the base address of the MTS version of CP/M is 4000H. This is the fundamental difference between the two versions of CP/M. By taking existing programs and translating CP/M related addresses up by 4000H these programs will run in the MTS CP/M environment.

Specifically, the following addresses must be modified:

(a) default file control block from 005CH - 007CH changed to 405CH - 407CH

(b) default 128 byte disk buffer from 0080H - 00FFH changed to 4080H - 40FFH

(c) default address of transient program area from 0100H changed to 4100H

2. PRT Program

A program called PRT has been written for the MTS CP/M operating system which provides a printer system command capability. PRT may be used in two ways.

First, if a user enters an I/O control P to CP/M the printer switch is activated thereby causing all information which is displayed on the terminal screen to also be displayed on the printer. Recall that the printer requires

an end of file character before information output to the printer is actually printed. After the user has issued the command:

PRT

the program outputs the end of file character thereby causing the printing of the latest information.

Second, PRT may be utilized to print any ASCII CP/M file. After the user enters the command:

PRT <filename>

the PRT program causes the hardcopy output of the file.

H. REFERENCES

1. Brown, K. J. and Bullock, D. R., A Shared Environment for Microcomputer System Development, M.S. Thesis, NPS, Monterey, CA, March 1977.

2. Carro, S. J. and Knouse, B. L., Implementation of an Operating System for a Shared Microcomputer Environment, M.S. Thesis, NPS, Monterey, CA, September 1977.

3. Digital Research, An Introduction to CP/M Features and Facilities, Pacific Grove, CA, 93950, 1976.

4. Digital Research, CP/M Assembler (ASM) User's Guide, Pacific Grove, CA, 93950, 1976.

5. Digital Research, CP/M Dynamic Debugging Tool (DDT), Pacific Grove, CA, 93950, 1976.

6. Digital Research, CP/M Interface Guide, Pacific Grove, CA, 93950, 1976.

7. Digital Research, CP/M Alteration Guide, Pacific Grove, CA, 93950, 1976.

8. Digital Research, ED: a Context Editor for the CP/M Disk System User's Manual, Pacific Grove, CA, 93950, 1975.

9. Intel Corp., 8080 Assembly Language Programming Manual, Santa Clara, CA, 95051, 1976.

10. Sycor Inc., Model 340/340D Intelligent Communications Terminal Operator's Manual, Nr TD34003-275, Ann Arbor, Michigan, Oct. 1974.

11. Sycor Inc., User's Guide to the 440 System, Nr UG4400-2, Ann Arbor, Michigan, November 1976.

12. Sycor Inc., 8080 Debug User's Guide, Spec 950706, Rev B, Ann Arbor, Michigan, October 1975.

```
/*********************************************************/
/*                GLOBAL IDENTIFIERS                     */
/*********************************************************/
/*                                                       */
/* THE FOLLOWING DECLARATIONS DEFINE SYSTEM IDENTI-      */
/* FIERS WHOSE SCOPE IS GLOBAL THROUGHOUT MTS. THESE     */
/* IDENTIFIERS MAY BE DIVIDED INTO THREE DISTINCT        */
/* GROUPS. THE FIRST GROUP INCLUDES ANY IDENTIFIER       */
/* CONSIDERED GLOBAL BECAUSE IT IS REFERENCED IN TWO     */
/* OR MORE MODULES OF THE MTS ML80 SOURCE PROGRAM. BY    */
/* INCLUDING THE DECLARATIONS FOR ALL SUCH IDENTIFIERS*/
/* IN A SINGLE MODULE, INTERMODULE LINKAGE IS GREATLY    */
/* SIMPLIFIED, AND THE SOURCE PROGRAMS'S READABILITY     */
/* AND CLARITY ARE IMPROVED.                             */
/*                                                       */
/* THE SECOND GROUP OF IDENTIFIERS INCLUDES THOSE        */
/* VARIABLES WHICH, TAKEN TOGETHER, DEFINE THE STATE     */
/* OF THE SYSTEM, I.E. THE SYSTEM STATE BLOCK. THE       */
/* CONCEPT OF SYSTEM STATE IS IMPORTANT IN MTS BECAUSE*/
/* THE SYCOR 440 HARDWARE ARCHITECTURE PROVIDES NO       */
/* PROTECTION AGAINST INADVERTENT OR MALICIOUS TAMP-     */
/* ERING WITH SYSTEM CODE BY USER PROGRAMS. TO MINI-     */
/* MIZE THE EFFECTS OF SYSTEM CRASHES CAUSED BY SUCH     */
/* TAMPERING MTS PROVIDES A LIMITED RECOVERY CAPABIL-    */
/* ITY. AFTER A TASK'S TIMESLICE EXPIRES, AND JUST       */
/* PRIOR TO INITIATING A NEW TASK, THE MTS MONITOR       */
/* COPIES THE CONTENTS OF THE SYSTEM STATE BLOCK TO A    */
/* FILE NAMED .MTSRCVR. IF A CRASH OCCURS WHILE THE      */
/* NEW TASK IS EXECUTING, RECOVERY MAY BE ACCOMPLISHED*/
/* BY REBOOTING MTS AND READING THE CONTENTS OF          */
/* .MTSRCVR BACK INTO THE SYSTEM STATE BLOCK. THE        */
/* TASK WHICH CAUSED THE CRASH IS THEN DELETED AND       */
/* NORMAL OPERATION CONTINUES.                           */
/*                                                       */
/* THE THIRD AND FINAL GROUP OF IDENTIFIERS INCLUDES     */
/* SYSTEM DATA ASSOCIATED WITH A PARTICULAR USER TASK.*/
/* SINCE THIS DATA IS ONLY USED WHEN ITS ASSOCIATED      */
/* TASK IS ACTIVE, THE SPACE REQUIRED FORMS A SYSTEM     */
/* AREA IN THE TASK'S SWAP IMAGE. THIS DATA IS SWAPPED*/
/* IN AND OUT ALONG WITH THE USER AREA OF THE SWAP       */
/* IMAGE.                                                */
/*                                                       */
/* THE THREE PRIMARY IDENTIFIER GROUPS DESCRIBED         */
/* ABOVE MAY ALSO BE SUBDIVIDED BASED ON USAGE AND       */
/* STORAGE ALLOCATION REQUIREMENTS. THE GROUP AND        */
/* SUBGROUP HEADINGS FOR DECLARATIONS IN THIS MODULE     */
/* ARE AS FOLLOWS:                                       */
/*                                                       */
/*       A.  GENERAL SYSTEM DECLARATIONS                 */
/*       B.  SYSTEM STATE BLOCK DECLARATIONS             */
/*           1.  SYSTEM CONTROL                          */
/*           2.  TASK CONTROL TABLE                      */
/*           3.  DISK MAP TABLE                          */
/*       C.  SYSTEM SWAP AREA DECLARATIONS               */
/*           1.  VIRTUAL DISK CONTROL BLOCK              */
/*           2.  SWAP STACK                              */
/*                                                       */
/* THE ORDER OF ALL DECLARATIONS IN THIS MODULE MUST     */
/* BE MAINTAINED TO PRODUCE A PROPERLY FORMATTED         */
/* OBJECT MODULE. IN THIS REGARD SPECIFICATION OF THE    */
/* INITIAL ATTRIBUTE IN A DECLARATION MUST BE CONSID-    */
/* ERED CAREFULLY SINCE THE ML80 COMPILER ALLOCATES      */
/* DIFFERENT AREAS OF MEMORY FOR INITIALIZED AND         */
/* UNINITIALIZED VARIABLES. SPECIAL PRECAUTIONS ARE      */
/* ALSO NECESSARY FOR LOCAL VARIABLES USED ONLY IN       */
/* SINGLE MODULES. THE ML80 LINK EDITOR IS FORCED TO     */
/* ALLOCATE SPACE FOR SUCH VARIABLES WITHIN THE          */
/* MODULE'S CODE AREA BY DECLARING EACH SUCH VARIABLE    */
```

```
/* WITH TYPE DATA. THIS TECHNIQUES IMPOSES A PENALTY  */
/* OF THREE BYTES PER DECLARATION FOR UNNECESSARY     */
/* JUMP INSTRUCTIONS, BUT THE SIMPLIFICATION OF       */
/* INTERMODULE LINKAGES MAKES THE TRADEOFF WORTHWHILE.*/
/*                                                    */
/******************************************************/
/******************************************************/


/******************************************************/
/************ GENERAL SYSTEM DECLARATIONS *************/
/******************************************************/


DECLARE PRT$STAT BYTE INITIAL (0);
    /* PRINTER STATUS BYTE */
DECLARE BUF$PTR(2) BYTE INITIAL (0,1);
    /* POINTER INTO PRINT BUFFER */
DECLARE RESERVED(13) BYTE INITIAL (0,0,0,0,0,0,0,0,0,0,
    0,0,0);
    /* RESERVED FOR CASSETTE AND ASYNC INTERFACE       */
DECLARE PARM(2) BYTE INITIAL (0,0);
DECLARE DISK BYTE INITIAL (0);
DECLARE DRIVE BYTE INITIAL (0);
DECLARE ERROR BYTE INITIAL (0);
DECLARE LOCK BYTE INITIAL (1);
    /* SYSTEM LOCK -                                   */
    /*     BIT 0: SWAP LOCK - MTS CODE EXECUTING       */
    /*     BITS 1-7:  (NOT USED)                       */
DECLARE TASK$TIMER BYTE INITIAL (0FFH);
    /* COUNTER RECORDING HOW MANY TIMER INCREMENTS     */
    /* (50MS) REMAIN IN TIMESLICE                      */
DECLARE SVC$STACK(20) BYTE INITIAL (0);
    /* SERVICE MODULE STACK */
DECLARE SYS$STACK(20) BYTE INITIAL (0);
    /* MONITOR MODULE STACK */
DECLARE MDBUF(512) BYTE INITIAL (0);
    /* MINI-DISK BUFFER - CONTAINS ONE SECTOR */
DECLARE MDSAD(2) BYTE INITIAL (0,0);
    /* SECTOR NUMBER  OF DATA CONTAINED IN MDBUF */


/******************************************************/
/*********** SYSTEM STATE BLOCK DECLARATIONS **********/
/******************************************************/


/******************* SYSTEM CONTROL *******************/

DECLARE TASK BYTE INITIAL (0);
    /* TERMINAL NR OF TASK CURRENTLY ALLOCATED     */
    /* THE CPU - RANGE 0-3                         */
DECLARE REC$FILE(2) BYTE INITIAL (0,0);
    /* MINI-DISK SECTOR NUMBER  OF .MTSRCVR */
DECLARE CNFG$FILE(2) BYTE INITIAL (0,0);
    /* MINI-DISK SECTOR NUMBER  OF .MTSCNFG */

/***************** TASK CONTROL TABLE *****************/
/* THE TCT CONTAINS INFORMATION ON THE STATE OF EACH  */
/* TASK AND DATA REQUIRED TO SUPPORT SWAPPING. EACH   */
/* VARIABLE CONTAINS FOUR ENTRIES - ONE FOR EACH OF   */
/* THE FOUR TERMINAL TASKS.                           */
/******************************************************/

DECLARE TCT$STATUS(4) BYTE INITIAL (0,0,0,0);
    /*     BIT 0: SIMULATE BOOTSTRAP DURING NEXT    */
    /*              TIMESLICE                       */
    /*     BIT 1: CALL MCP DURING NEXT TIMESLICE    */
    /*     BIT 2:  (NOT USED)                       */
    /*     BIT 3:  (NOT USED)                       */
    /*     BIT 4: RST 7 DURING NEXT TIMESLICE       */
    /*     BIT 5: BLOCKED FOR TERMINAL I/O          */
    /*     BIT 6: CURRENT SWAP IMAGE RESIDES ON     */
```

```
    /*          MINI-DISK                                 */
    /*    BIT 7: CURRENT SWAP IMAGE IN MEMORY             */
DECLARE TCT$DM(32) BYTE INITIAL (0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0);
    /* DRIVE MAP - POINTERS TO VIRTUAL DISK ASSOC-   */
    /* IATED WITH EACH VIRTUAL DRIVE. BYTES 0-7      */
    /* CORRESPOND RESPECTIVELY TO DRIVES A-H FOR     */
    /* EACH 8 BYTE ENTRY.                            */
    /*    BITS 0-4: DISK NR - RANGE 0-31             */
    /*    BIT 5:  (NOT USED)                         */
    /*    BIT 6: READ ONLY FLAG                      */
    /*    BIT 7: IN USE FLAG                         */
DECLARE TCT$SIZE(4) BYTE INITIAL (32,32,32,32);
    /* SIZE OF SWAP IMAGE EXPRESSED IN NUMBER */
    /* OF 512 BYTE MINI-DISK SECTORS          */
DECLARE TCT$BOE(8) BYTE INITIAL (0,0,0,0,0,0,0,0);
DECLARE TCT$EOE(8) BYTE INITIAL (0,0,0,0,0,0,0,0);
    /* MINI-DISK SECTOR NUMBER  FOR EACH SWAP FILE */
    /* - INITIALIZED WHEN MTS LOADED               */

/****************** DISK MAP TABLE ******************/
/* THE DMT CONTAINS INFORMATION ON THE STATUS, PRO-   */
/* TECTION, AND LOCATION ON THE MINI-DISK OF ALL VIR- */
/* TUAL FLOPPY DISKS. EACH VARIABLE CONTAINS 32       */
/* ENTRIES - ONE FOR EACH POTENTIAL DISK NR. THE      */
/* ENTIRE TABLE IS LOADED AND VERIFIED DURING MTS     */
/* INITIALIZATION.                                    */
/*****************************************************/

DECLARE DMT$FLAG(32) BYTE INITIAL (0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
    /*    BIT 0: DISK EXISTS                          */
    /*    BIT 1: IN USE                               */
    /*    BIT 2: PROTECTED - KEY REQUIRED             */
    /*    BIT 3: RESTRICTED TO READ ONLY W/O KEY      */
    /*    BITS 4-7:  (NOT USED)                       */
DECLARE DMT$BOE(64) BYTE INITIAL (0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0);
    /* MINI-DISK SECTOR NUMBER  FOR BEGINNING */
    /* OF EXTENT                              */
DECLARE DMT$EOE(64) BYTE INITIAL (0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0);
    /* MINI-DISK SECTOR NUMBER  FOR END */
    /* OF EXTENT                        */
DECLARE DMT$KEY(128) BYTE INITIAL (20H);
    /* ONE TO FOUR ASCII CHAR PROTECTION KEY */


/*****************************************************/
/************ SYSTEM SWAP AREA DECLARATIONS ************/
/*****************************************************/


/************* VIRTUAL DISK CONTROL BLOCK *************/
/* EACH USER TASK HAS AVAILABLE 8 VIRTUAL DRIVES WHICH*/
/* MAY BE SELECTED TO ACCESS THE ATTACHED VIRTUAL     */
/* DISK. FOR EACH USER IT IS NECESSARY TO RECORD      */
/* WHICH DRIVE IS CURRENTLY ACTIVE, AND ADDITIONAL    */
/* DATA NEEDED TO MAP A VIRTUAL DISK ACCESS INTO A    */
/* PHYSICAL MINI-DISK ACCESS. ALL THIS INFORMATION IS */
/* MAINTAINED IN THE VDC BLOCK. THE VDC BLOCK ASSOC-  */
/* IATED WITH EACH TASK IS CONTAINED IN THAT TASKS    */
/* SWAP FILE IN A SPECIAL AREA RESERVED FOR MTS SYS-  */
/* TEM USE. THIS MEANS THAT ONLY ONE OF THE FOUR VDC  */
/* BLOCKS MAINTAINED BY THE SYSTEM IS EVER RESIDENT   */
/* IN MEMORY AT ANY ONE TIME.                         */
/*****************************************************/
```

```
DECLARE VDC$DRIVE BYTE;
    /*     BITS 0-2: DRIVE NR FOR DRIVE CURRENTLY    */
    /*                  SELECTED                     */
    /*     BITS 3-5:  (NOT USED)                     */
    /*     BIT 6: READ ONLY FLAG                     */
    /*     BIT 7: MODIFICATION FLAG - SET WHEN CON-  */
    /*                TENTS OF BUFFER MODIFIED       */
DECLARE VDC$BOE(2) BYTE;
    /* MINI-DISK SECTOR NUMBER  FOR BOE OF VIRTUAL */
    /* DISK CURRENTLY ATTACHED TO SELECTED DRIVE   */
DECLARE VDC$EOE(2) BYTE;
    /* MINI-DISK SECTOR NUMBER  FOR EOE OF VIRTUAL */
    /* DISK CURRENTLY ATTACHED TO SELECTED DRIVE   */
DECLARE VDC$SECTOR BYTE;
    /* VIRTUAL DISK SECTOR  NR FOR SUBSEQUENT */
    /* ACCESSES - RANGE 1-26                  */
DECLARE VDC$TRACK BYTE;
    /* VIRTUAL DISK TRACK NR FOR SUBSEQUENT */
    /* ACCESSES - RANGE 0-76                */
DECLARE VDC$DMA(2) BYTE;
    /* MEMORY ADDRESS OF 128 BYTE DMA BUFFER */
    /* FOR SUBSEQUENT VIRTUAL DISK ACCESSES  */
DECLARE PRT$CNTRL BYTE INITIAL (0);
    /*      BITS 0,1:    TASK        */
    /*      BITS 2-5:    NOT USED    */
    /*      BIT  6:      PRINT BIT   */
    /*      BIT  7:      IN USE BIT  */
DECLARE PRT$BOE(2) BYTE INITIAL (0,0);
    /* MINIDISK SECTOR # OF MTSPRT BOE  */
DECLARE PRT$EOE(2) BYTE INITIAL (0,0);
    /* MINIDISK SECTOR # OF MTSPRT EOE  */
DECLARE PRT$SEC(2) BYTE INITIAL (0,0);
    /* SECTOR # OF CURRENT SECTOR IN PRINT BUFFER */
DECLARE SPARES(9) BYTE INITIAL (0,0,0,0,0,0,0,0,0);
    /* RESERVED CASSETTE AND ASYNC INTERFACE      */

/********************** SWAP STACK **********************/
/* EACH TIME A TASK IS SWAPPED OUT THE CURRENT OPER-    */
/* ATING ENVIRONMENT, I.E. PSW, BC, DE, HL, AND SP,     */
/* MUST BE SAVED IN A KNOWN AREA SO THAT IT CAN BE      */
/* QUICKLY RESTORED WHEN THE TASK IS SWAPPED BACK IN.   */
/* MTS USES A STACK IN THE SYSTEM AREA OF THE SWAP      */
/* IMAGE TO HOLD THE ENVIRONMENT WHEN A TASK IS         */
/* INACTIVE.                                            */
/********************************************************/

DECLARE SWAP$STACK(10) BYTE;
    /* AREA IN WHICH USER ENVIRONMENT IS */
    /* SAVED WHEN TASK IS SWAPPED OUT    */

EOF
```

```
/*********************************************************/
/*                 INTERRUPT MODULE                      */
/*********************************************************/
/*                                                       */
/*                                                       */
/* ALL HARDWARE INTERRUPTS ON THE SYCOR 440 SYSTEM       */
/* CAUSE THE EXECUTION OF A RST 1 INSTRUCTION. THIS      */
/* INSTRUCTION BEHAVES LIKE A CALL TO LOCATION 0008H,    */
/* I.E. THE PC VALUE IS STACKED, AND CONTROL TRANS-      */
/* FERRED TO LOCATION 0008H. DUE TO THIS HARDWARE        */
/* CHARACTERISTIC, THE USER MUST ENSURE THAT ANY USER    */
/* DEFINED STACKS ARE AT LEAST FOUR BYTES LARGER THAN    */
/* THE MAXIMUM SIZE REQUIRED BY THE USER'S OWN CODE.     */
/* SINCE ALL PERIPHERAL DEVICES CAUSE EXECUTION OF       */
/* THE SAME INTERRUPT INSTRUCTION, SOME MEANS MUST BE    */
/* AVAILABLE TO DISTINQUISH BETWEEN DEVICES WHENEVER     */
/* AN INTERRUPT OCCURS. THE SYCOR 440 SOLVES THIS        */
/* PROBLEM BY DEFINING AN INTERRUPT LEVEL FOR EACH       */
/* DIFFERENT DEVICE. THERE ARE 17 INTERRUPT LEVELS       */
/* WITH VALUES RANGING FROM 0 TO 16. A HIGHER NUMERIC    */
/* VALUE ALSO IMPLIES A HIGHER PRIORITY FOR THE          */
/* ASSOCIATED DEVICE. WHEN AN INTERRUPT OCCURS THE       */
/* LEVEL IS AVAILABLE ON INPUT LATCH 0. SIMULTANEOUS     */
/* INTERRUPTS WILL BE INPUT SEQUENTIALLY IN PRIORITY,    */
/* I.E. DESCENDING, SEQUENCE BY LEVEL NUMBER. WHEN       */
/* THE LEVEL READS ZERO ALL PENDING INTERRUPTS HAVE      */
/* BEEN PROCESSED. THE INTERRUPT LEVEL ASSIGNMENTS       */
/* WHICH APPLY TO THE CURRENT NPS SYCOR 440 HARDWARE     */
/* CONFIGURATION ARE AS FOLLOWS:                         */
/*                                                       */
/*           LEVEL         DEVICE                        */
/*           -----         ------                        */
/*            16           DEBUGGER                      */
/*            15           POWER FAIL                    */
/*            14           PARITY CONTROL                */
/*            11           ASYNC COMM                    */
/*            10           TERMINAL GROUP 0              */
/*             8           TIMER                         */
/*             6           PRINTER 0                     */
/*             2           FLOPPY DISK                   */
/*             1           CASSETTE                      */
/*                                                       */
/* THIS MODULE CONTAINS THE CODE USED BY MTS TO PRO-     */
/* CESS INTERRUPTS. THIS CODE CONSISTS OF AN INTER-      */
/* RUPT CONTROLLER PLUS A SET OF INTERRUPT HANDLER       */
/* ROUTINES - ONE ROUTINE FOR EACH DEVICE. THE INTER-    */
/* RUPT CONTROLLER SAVES THE CURRENT ENVIRONMENT,        */
/* IDENTIFIES THE INTERRUPT LEVEL, CALLS THE APPROP-     */
/* RIATE HANDLER ROUTINE, AND THEN RESTORES THE          */
/* ENVIRONMENT BEFORE RETURNING TO THE INTERRUPTED       */
/* PROGRAM. THE HANDLER ROUTINES ARE TAILORED TO THE     */
/* SPECIFIC REQUIREMENTS OF DIFFERENT DEVICES. IN        */
/* ORDER TO UTILIZE THE CODE CONTAINED IN THE INTER-     */
/* RUPT MODULE IT IS NECESSARY FOR THE MTS INITIAL-      */
/* IZATION ROUTINE TO LOAD A JUMP TO THE INTERRUPT       */
/* CONTROLLER IN MEMORY LOCATIONS 0008-000AH.            */
/*                                                       */
/*                                                       */
/*********************************************************/
/*********************************************************/


/************* INTERMODULE LINKAGE MACROS *************/

DECLARE GLOB1 COMMON;
[INT MB M2B TB]
[MB:=0300H]  [TB:=1000H]  [M2B:=0600H]
[MACRO MONITOR '[HEX MB + 0AAH]']
[MACRO MOVBUF '[HEX M2B + 41H]']
[MACRO READ$PRT$BUF '[HEX M2B + 9EH]']
[MACRO TERM$INPUT$CTRL '[HEX TB + 735H]']
[MACRO BLINK$CURSOR '[HEX TB + 803H]']
[MACRO MTS$MSG '[HEX TB + 837H]']
```

```
[MACRO PRT$STAT 'GLOB1(0)']
[MACRO BUF$PTR 'GLOB1(1)']
[MACRO LOCK 'GLOB1(15H)']
[MACRO TASK$TIMER 'GLOB1(16H)']
[MACRO TASK 'GLOB1(241H)']
[MACRO PRT$CNTRL 'GLOB1(39EH)']
[MACRO SWAP$STACK 'GLOB1(3B7H)']

/************ INTERRUPT PROCESSING MACROS *************/

[MACRO LEVEL 'IN(0)']
[MACRO DEBUG$LATCH '0FFH']
[MACRO CSST$LATCH '85H']
[MACRO PRINTER$LATCH '8AH']
[MACRO PRT$CHAR 'OUT(8AH)=A']
[MACRO PRT$RDY '0B0H']
[MACRO CNTRL$Z '1AH']
[MACRO TIMER$LATCH '02H']
[MACRO TERMINAL$LATCH '3EH']
[MACRO MATRIX$LATCH '3FH']
[MACRO INT$PENDING '(A=[LEVEL]\A) !ZERO']
[MACRO DEBUG$INT$PENDING '(A=[LEVEL]; A::16) ZERO']

/**************** MODULE DECLARATIONS *****************/

[INT TOP BLINK EP] [BLINK:=4] [TOP:=30] [EP:=TOP-10]
DECLARE INT$STACK DATA (0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
DECLARE BLINK$TIMER DATA (1);
DECLARE SAVHL DATA (0,0);
DECLARE TN DATA (0);

/********************** PROCEDURES *********************/

DUMMY$HDLR:  PROCEDURE;
/*******************************************************/
/* THIS PROCEDURE PROVIDES A COMMON EMPTY INTERRUPT    */
/* HANDLER FOR THOSE INTERRUPT LEVELS WHICH SHOULD     */
/* NEVER OCCUR WITH THE CURRENT NPS SYCOR 440 HARD-    */
/* WARE CONFIGURATION. ITS ONLY ACTION IS AN IMMEDI-   */
/* ATE RETURN.                                         */
/* CALLED BY: INTERRUPT$CONTROLLER                     */
/*******************************************************/
    RETURN;
    END DUMMY$HLDR;

DEBUG$HDLR:  PROCEDURE;
/*******************************************************/
/* THIS HANDLER IS INCLUDED TO ALLOW USAGE OF THE      */
/* SYCOR 340B DEBUGGER IN THE SOFT DEBUG MODE WHEN     */
/* RUNNING UNDER MTS. IT WAS DISCOVERED DURING DEVEL-  */
/* OPMENT AND TESTING THAT THE DEBUGGER IS LARGELY     */
/* UNRELIABLE IN THE SOFT MODE. THIS IS APPARENTLY     */
/* DUE TO HARDWARE CHANGES IN THE 440 SYSTEM MADE      */
/* AFTER THE DEBUGGER WAS DESIGNED.                    */
/* CALLED BY: INTERRUPT$CONTROLLER                     */
/*******************************************************/
    /* DISPLAY ENVIRONMENT ON DEBUGGER */
    HL=2+SP;
    M(6H)=(A=L); M(7H)=(A=H);
    /* ACKNOWLEDGE DEBUG INTERRUPT */
    OUT([DEBUG$LATCH])=(A=1);
    /* CPU IDLES WHILE WAITING FOR DEBUGGER TO */
    /* INITIATE RESUMPTION OF EXECUTION        */
    DO WHILE [DEBUG$INT$PENDING];
        END;
    END DEBUG$HDLR;

CASSETTE$HDLR:  PROCEDURE;
    OUT([CSST$LATCH])=(A=10H);
    END CASSETTE$HDLR;

PRINTER$HDLR:  PROCEDURE;
```

```
/*********************************************************/
/* THE PRINTER INTERRUPT HANDLER INPUTS THE PRINTER      */
/* STATUS.  IF THE CONTROL BYTE INDICATES THAT NO TASK IS */
/* CURRENTLY ALLOCATED THE PRINTER, NO ACTION IS TAKEN.  */
/* HOWEVER, IF A TASK IS ALLOCATED THE PRINTER AND THE   */
/* DEVICE IS NOT READY THEN THE MESSAGE 'PRINTER NOT RDY' */
/* IS DISPLAYED AT THE USER'S TERMINAL; OTHERWISE A      */
/* CHARACTER FROM THE PRINT BUFFER LOCATED AT 100H IS    */
/* OUTPUT TO THE PRINTER.  THIS PROCESS CONTINUES UNTIL  */
/* AN END OF FILE CHARACTER HAS BEEN OUTPUT TO THE       */
/* PRINTER AT WHICH TIME THE DEVICE IS RELEASED FOR USE  */
/* BY ANOTHER TASK.                                      */
/* CALLED BY: INTERRUPT$CONTROLLER                       */
/*********************************************************/
     A=IN([PRINTER$LATCH]);
     [PRT$STAT]=A;
     OUT([PRINTER$LATCH])=(A=80H); /* CLEAR PRINTER INTERUPT */
     IF (A=[PRT$STAT]; A::[PRT$RDY]) !ZERO THEN
     DO;  /* PRINTER NOT READY */
          IF (A=[PRT$CNTRL]; A::0) !ZERO THEN
          DO;  /* GET TASK # FOR ERROR MESSAGE */
               TN=(A=[TASK]);
               [TASK]=(A=[PRT$CNTRL] & 03H);
               E=7;  CALL [MTS$MSG];
               [TASK]=(A=TN);
               [PRT$CNTRL]=(A=0); /* RESET PRINT CONTROL */
               RETURN;
          END;
     END
     ELSE
     DO; /* PRINTER IS READY FOR CHARACTER OUTPUT */
          IF (A=<[PRT$CNTRL]; A=<A) CY THEN
          DO;
               HL=[BUF$PTR];
               IF (A=3; A::H) ZERO & (A=0; A::L) ZERO THEN
               DO; /* END OF PRINT BUFFER, GET NEXT SECTOR */
                    CALL [READ$PRT$BUF];
                    [BUF$PTR]=(HL=100H);
               END;
               IF (A=M(HL);  A::[CNTRL$Z]) !ZERO THEN
               DO; /* PRINT OUT CHARACTER */
                    [PRT$CHAR];
                    [BUF$PTR]=(HL=[BUF$PTR], +1);
                    /* BUF$PTR INCREMENTED */
                    RETURN;
               END
               ELSE
               /* EOF ENCOUNTERED, RESET PRINTER */
               [PRT$CNTRL]=(A=0);
          END;
     END;
     END PRINTER$HDLR;

TIMER$HDLR:  PROCEDURE;
/*******************************************************/
/* THE TIMER INTERRUPT HANDLER MANAGES THE TWO FUNC-   */
/* TIONS OF MTS WHICH OCCUR AT PERIODIC INTERVALS:     */
/* BLINKING THE TERMINAL CURSORS AND RETURNING CONTROL*/
/* TO THE SYSTEM WHEN A TASK'S TIMESLICE EXPIRES. IN   */
/* ORDER TO KEEP TRACK OF THE TWO INTERVALS INVOLVED, */
/* THE PROCEDURE MAINTAINS TWO COUNTERS. THESE COUN-   */
/* TERS ARE EACH SET TO AN INITIAL VALUE AND THEN      */
/* DECREMENTED EACH TIME A TIMER INTERRUPT OCCURS.     */
/* THE ACTUAL VALUE CONTAINED IN EITHER COUNTER AT ANY*/
/* INSTANT REPRESENTS THE TIME REMAINING IN THE INTER-*/
/* VAL IN MULTIPLES OF 50MS SINCE THIS IS THE FIXED    */
/* INTERVAL BETWEEN TIMER INTERRUPTS. WHEN THE TASK$   */
/* TIMER COUNTER HAS BEEN DECREMENTED TO ZERO, CONTROL*/
/* IS TRANSFERRED TO THE MONITOR WHERE THE CURRENT     */
/* TASK IS SWAPPED OUT AND A NEW TASK SWAPPED IN.      */
/* WHEN THE BLINK$TIMER COUNTER REACHES ZERO THE       */
/* BLINK$CURSOR PROCEDURE IS CALLED. IN EITHER CASE    */
/* THE TIMER HANDLER RESETS THE COUNTER TO ITS INIT-   */
```

```
/* IAL VALUE AND CONTINUES.                                 */
/* CALLED BY: INTERRUPT$CONTROLLER                          */
/************************************************************/
    [TASK$TIMER]=(A=[TASK$TIMER]-1);
    BLINK$TIMER=(A=BLINK$TIMER-1);
    IF (A::0) ZERO THEN  /* BLINK INTERVAL EXPIRED */
        DO;
        CALL [BLINK$CURSOR];
        BLINK$TIMER=(A=[BLINK]);
        END;
    OUT([TIMER$LATCH])=(A=0);  /* RESET TIMER */
    IF (A=[TASK$TIMER]; A::0) ZERO THEN
        DO;  /* TIMESLICE EXPIRED */
        IF (A=>[LOCK]) !CY THEN
            DO;  /* SWAPPING UNLOCKED */
            BC=10; DE=.INT$STACK([EP]);
            HL=.[SWAP$STACK];
            CALL [MOVBUF];
            ENABLE;
            GOTO [MONITOR];
            END
        ELSE [TASK$TIMER]=(A=1);
        END;
    END TIMER$HDLR;

TERMINAL$HDLR:  PROCEDURE;
/************************************************************/
/* THIS PROCEDURE PROCESSES THE INTERRUPT GENERATED         */
/* FROM ANY KEY DEPRESSION AT ANY OF THE TERMINALS.         */
/* IT GETS THE TERMINAL IDENTITY AND THE KEYBOARD           */
/* MATRIX CODE AND THEN CALLS TERMINAL$INPUT$CONTROL        */
/* TO PROCESS THE KEY.                                      */
/* OUTPUT: C - MATRIX CODE                                  */
/*         E - TERMINAL NUMBER                              */
/* CALLED BY: INTERRUPT$CONTROLLER                          */
/************************************************************/
    /* READ TERMINAL IDENTITY */
    E=(A=IN([TERMINAL$LATCH]) & 03);
    /* WRITE TERMINAL NUMBER BACK OUT TO CAUSE */
    /* THE APPROPRIATE KEYBOARD DATA REGISTER  */
    /* TO BE SELECTED FOR READING.             */
    OUT([TERMINAL$LATCH])=A;
    /* READ THE KEYBOARD MATRIX CODE */
    C=(A=IN([MATRIX$LATCH]));
    /* PROCESS KEY */
    CALL [TERM$INPUT$CTRL];
    END TERMINAL$HDLR;


INTERRUPT$CONTROLLER:  /* MAIN ENTRY INTO INTMOD */
    /* SAVE CURRENT VALUE OF STACK POINTER AND */
    /* ALL REGISTERS IN INT$STACK              */
    SAVHL=HL; STACK=PSW;
    HL=2+SP; PSW=STACK;
    SP=.INT$STACK([TOP]);
    STACK=HL;  /* PUSH CURRENT STACK PTR */
    HL=SAVHL;
    STACK=HL;  /* PUSH ORIGINAL CONTENTS OF HL */
    STACK=DE; STACK=BC; STACK=PSW;
    DO WHILE [INT$PENDING];
        H=(A=0); L=(A=[LEVEL]);
        DO CASE HL;
         /*  0 */  CALL DUMMY$HDLR;
         /*  1 */  CALL CASSETTE$HDLR;
         /*  2 */  CALL DUMMY$HDLR;
         /*  3 */  CALL DUMMY$HDLR;
         /*  4 */  CALL DUMMY$HDLR;
         /*  5 */  CALL DUMMY$HDLR;
         /*  6 */  CALL PRINTER$HDLR;
         /*  7 */  CALL DUMMY$HDLR;
         /*  8 */  CALL TIMER$HDLR;
         /*  9 */  CALL DUMMY$HDLR;
         /* 10 */  CALL TERMINAL$HDLR;
```

```
        /* 11 */  CALL DUMMY$HDLR;
        /* 12 */  CALL DUMMY$HDLR;
        /* 13 */  CALL DUMMY$HDLR;
        /* 14 */  CALL DUMMY$HDLR;
        /* 15 */  CALL DUMMY$HDLR;
        /* 16 */  CALL DEBUG$HDLR;
      END; /* CASE */
   END; /* WHILE */
   /* RESTORE ORIGINAL VALUE OF STACK POINTER AND */
   /* ALL REGISTERS FROM INT$STACK                */
   PSW=STACK; BC=STACK; DE=STACK; HL=STACK;
   SAVHL=HL; HL=STACK;
   SP=HL; HL=SAVHL;
   ENABLE;
   RETURN;
   /* END INTERRUPT$CONTROLLER */

EOF
```

```
/*********************************************************/
/*                 MONITOR MODULE                        */
/*********************************************************/
/*                                                       */
/* THE MONITOR MODULE CONTAINS THOSE FUNCTIONS OF MTS    */
/* WHICH DEAL WITH PROCESSOR MANAGEMENT. SUCH FUNC-      */
/* TIONS INCLUDE THE INITIAL PROGRAM LOAD, SYSTEM        */
/* RECOVERY, SCHEDULING, CPU ALLOCATION, AND SWAPPING.   */
/* THE MODULE IS DIVIDED INTO THREE BASIC SUBMODULES.    */
/*                                                       */
/*  (1) UTILITY PROCEDURES                               */
/*      THIS SUBMODULE CONTAINS GENERAL PURPOSE          */
/*      UTILITY PROCEDURES WHICH PERFORM OPERATIONS      */
/*      FREQUENTLY REQUIRED IN THE MONITOR, INTERRUPT    */
/*      AND SERVICE MODULES.                             */
/*          * INDEX               * PUT                  */
/*          * INDEX2              * GET                  */
/*          * INDEX4              * MOVBUF               */
/*          * INDEX8              * MINI$DISK            */
/*          * READ$PRT$BUF                               */
/*                                                       */
/*  (2) TASK MANAGEMENT                                  */
/*      THIS SUBMODULE CONTAINS THE SCHEDULING, CPU      */
/*      ALLOCATION, AND SWAPPING PROCEDURES. IT ALSO     */
/*      INCORPORATES THE MECHANISM FOR RECORDING THE     */
/*      SYSTEM STATE BLOCK WHEN THE SYSTEM STATE         */
/*      CHANGES, THUS MAKING RECOVERY POSSIBLE.          */
/*      CONTROL PASSES TO THE TASK MANAGEMENT            */
/*      SUBMODULE AFTER MTS HAS BEEN INITIALIZED BY      */
/*      THE IPL SUBMODULE.                               */
/*          * MONITOR             * SWAP                 */
/*          * BUMP$TASK           * WRITE$REC            */
/*          * BOOTSTRAP           * RESUME$EXECUTION     */
/*                                                       */
/*  (3) INITIAL PROGRAM LOAD                             */
/*      THIS SUBMODULE CONTAINS ALL PROCEDURES WHICH     */
/*      DEAL WITH THE LOADING PROCESS AFTER THE MTS      */
/*      OBJECT MODULE HAS BEEN LOADED INTO MEMORY BY     */
/*      THE SYCOR SYSTEM LOADER. THE PRIMARY FUNCTION    */
/*      OF THE IPL SUBMODULE IS SYSTEM INITIALIZATION    */
/*      OR RECOVERY, AS REQUIRED. IN ORDER TO MINI-      */
/*      MIZE THE MEMORY REQUIREMENT OF THE RESIDENT      */
/*      MTS CODE, THIS SUBMODULE WAS WRITTEN AS A        */
/*      STANDALONE PROGRAM LOADED INTO THE USER SWAP     */
/*      AREA. ONCE IPL IS COMPLETE THE PROGRAM MAY       */
/*      BE OVERLAYED BY USER PROGRAMS.                   */
/*          * ABORT$IPL           * READ$DIRECTORY       */
/*          * SEARCH$DIRECTORY    * RECOVER              */
/*          * INITIALIZE          * MTS$IPL              */
/*          * RECOVER$STATUS$LINE                        */
/*                                                       */
/* THE MONITOR MODULE REQUIRES ACCESS TO SEVERAL         */
/* FILES WHICH RESIDE ON THE MINI-DISK IN SYCOR          */
/* FORMAT. THESE FILES AND THEIR FUNCTION ARE:           */
/*                                                       */
/*  (1) .MTSSWP0,.MTSSWP1,.MTSSWP2,.MTSSWP3              */
/*      ASSOCIATED WITH EACH OF THE FOUR TERMINAL        */
/*      TASKS IS A FILE USED TO STORE A CORE IMAGE       */
/*      OF THE TASK WHEN IT IS INACTIVE OR BLOCKED.      */
/*      THIS SWAP IMAGE CONSISTS OF A SYSTEM AREA        */
/*      WHERE THE ENVIRONMENT AND VIRTUAL DEVICE         */
/*      CONTROL BLOCK IS STORED, AND A USER AREA         */
/*      CONTAINING THE USER PROGRAM'S MEMORY IMAGE.      */
/*      THE MAXIMUM SWAP IMAGE SIZE IS 48K BYTES.        */
/*      THEREFORE, EACH SWAP FILE SHOULD NORMALLY BE     */
/*      96 MINI-DISK SECTORS LONG. IN THE EVENT THAT     */
/*      MINI-DISK SPACE IS LIMITED AND THE ENTIRE 48K    */
/*      IS NOT REQUIRED FOR USER PROGRAMS, MTS WILL      */
/*      AUTOMATICALLY ADJUST TO ANY FILE SIZE GREATER    */
/*      THAN 16K (32 SECTORS). THE FOLLOWING SYCOR       */
/*      COMMAND MAY BE USED TO CREATE A SWAP FILE:       */
/*              CREATE <FILENAME> N=96                   */
/*      THE SYCOR SYSTEM DOES NOT ALLOW DYNAMIC          */
```

```
/*       CHANGES IN FILE SIZE OR ATTRIBUTES; THUS, IN  */
/*       ORDER TO CHANGE THE FILE SIZE, THE FILE MUST  */
/*       FIRST BE DELETED (DELETE <FILENAME>) AND THEN */
/*       RECREATED WITH THE DESIRED SIZE. IF SWAP      */
/*       FILES SMALLER THAN 48K ARE DESIRED, THE VALUE */
/*       OF N IN THE CREATE COMMAND STRING SHOULD BE   */
/*       TWO TIMES THE REQUIRED PROGRAM SPACE IN       */
/*       KILOBYTES.                                    */
/*       THE FOUR SWAP FILES ARE ONLY REQUIRED WHEN    */
/*       MTS IS ACTUALLY RUNNING, OR BETWEEN RUNS IF   */
/*       RECOVERY WILL BE REQUESTED.                   */
/*                                                     */
/*   (2) .MTSCNFG                                      */
/*       THE VIRTUAL FLOPPY DISK CONFIGURATION FILE    */
/*       PROVIDES THE MAPPING BETWEEN THE VIRTUAL DISK */
/*       NUMBER (0-31) AND THE NAME OF A SYCOR FILE    */
/*       WHICH CONTAINS A FLOPPY DISK IMAGE. THE CON-  */
/*       FIGURATION FILE ALSO CONTAINS THE PROTECTION  */
/*       ATTRIBUTES OF THE VIRTUAL DISKS.  A           */
/*       PHYSICAL FLOPPY DISK HAS A FIXED CAPACITY OF  */
/*       256K BYTES, WHERE AN MTS VIRTUAL FLOPPY DISK  */
/*       MAY HAVE ANY CONVIENT SIZE UP TO 256K BYTES.  */
/*                           THE SYSTEM ASSUMES THAT THE */
/*       DISK IMAGE IS STORED SEQUENTIALLY ON THE      */
/*       MINI-DISK STARTING WITH TRACK 0 SECTOR 1 AND  */
/*       PROCEEDING THROUGH 26 SECTORS OF TRACK 0 TO   */
/*       TRACK 1 SECTOR 0, ETC. SPECIFYING A VIRTUAL   */
/*       DISK FILE SMALLER THAN 256K BYTES MEANS THAT  */
/*       FEWER THAN 77 TRACKS WILL BE ADDRESSABLE.     */
/*       THE CONFIGRATION FILE CONTAINS 32 THIRTEEN    */
/*       BYTE RECORDS IN THE FOLLOWING FORMAT:         */
/*               ----------------------                */
/*               | FILENAME | KEY | P |                */
/*               ----------------------                */
/*                0        7 8  11  12                 */
/*       WHERE 'FILENAME' IS THE 0-8 BYTE NAME OF THE  */
/*       VIRTUAL DISK FILE AS IT APPEARS IN THE SYCOR  */
/*       DIRECTORY; 'KEY' IS A 0-4 BYTE PROTECTION     */
/*       KEY; AND 'P' IS THE PROTECTION ATTRIBUTE OF   */
/*       THE DISK, I.E. 'P' FOR READ/WRITE PROTECTION, */
/*       'R' FOR WRITE PROTECTION ONLY, AND BLANK FOR  */
/*       NO PROTECTION.                                */
/*       THE CONFIGURATION FILE WILL BE UPDATED BY THE */
/*       MTS SYSTEM COMMANDS PROTECT, UNPROTECT, AND   */
/*       RESTRICT.  IN                                 */
/*       THE EVENT THAT .MTSCNFG IS ERRONEOUSLY        */
/*       DELETED, THE FILE MAY BE RECREATED BY USING   */
/*       THE SYCOR COMMAMDS                            */
/*               CREATE .MTSCNFG N=1                   */
/*               RUN BATCH 1=/CSST 3=.MTSCNFG          */
/*       WITH THE CASSETTE LABELED ".MTSCNFG"          */
/*       MOUNTED ON THE CASSETTE DRIVE.                */
/*                                                     */
/*   (3) .MTSRCVR                                      */
/*       THE RECOVERY FILE CONTAINS A COPY OF THE      */
/*       SYSTEM STATE BLOCK AS OF THE LAST SWAP. THE   */
/*       FILE IS ONLY REQUIRED WHEN MTS IS ACTUALLY    */
/*       RUNNING, OR BETWEEN RUNS IF RECOVERY WILL BE  */
/*       REQUESTED. THE FOLLOWING SYCOR COMMAND IS     */
/*       USED TO CREATE THE FILE:                      */
/*               CREATE .MTSRCVR N=2                   */
/*                                                     */
/*******************************************************/
/*******************************************************/


/*******************************************************/
/****************** TASK MANAGEMENT *******************/
/*******************************************************/
```

```
/************* INTERMODULE LINKAGE MACROS *************/

[INT TB M2B GB] [TB:=1000H] [M2B:=0600H] [GB:=0]
[MACRO MCP '1A00H']
[MACRO MTS '1F00H']
[MACRO INDEX '[HEX M2B + 3H]']
[MACRO INDEX2 '[HEX M2B + 0DH]']
[MACRO INDEX8 '[HEX M2B + 24H]']
[MACRO GET '[HEX M2B + 38H]']
[MACRO MINI$DISK '[HEX M2B + 54H]']
[MACRO MTS$MSG '[HEX TB + 837H]']
[MACRO READ$TERMINAL '[HEX TB + 8DCH]']
[MACRO GET$TERM$STATUS '[HEX TB + 2F9H]']
[MACRO LOCK 'M([HEX GB + 3C54H])']
[MACRO TASK$TIMER 'M([HEX GB + 3C55H])']
[MACRO SYS$STACK$TOP '[HEX GB + 3C6AH + 20]']
[MACRO TASK$ADDR '[HEX GB + 3E80H]']
[MACRO TASK 'M([TASK$ADDR])']
[MACRO REC$FILE '[HEX GB + 3E81H]']
[MACRO TCT$STATUS '[HEX GB + 3E85H]']
[MACRO TCT$DM '[HEX GB + 3E89H]']
[MACRO TCT$SIZE '[HEX GB + 3EA9H]']
[MACRO TCT$BOE '[HEX GB + 3EADH]']
[MACRO DMT$BOE '[HEX GB + 3EDDH]']
[MACRO VDC$DRIVE '[HEX GB + 3FEDH]']
[MACRO SWAP$STACK '[HEX GB + 3FF6H]']
[MACRO SWAP$STACK1 'M([HEX GB + 3FF7H])']
/* .MDBUF = 3C7EH USED IN WRITE$REC      */

/*************** GENERAL PURPOSE MACROS ***************/

[INT MEM$BASE TOP TIMESLICE]
[MEM$BASE:=4000H]  [TIMESLICE:=4]  [TOP:=20]
[MACRO READ '1']
[MACRO WRITE '2']
[MACRO BUMP '-2']
[MACRO DISK$ERROR '(A::0) !ZERO']
[MACRO HARDWARE$ERROR '8']
[MACRO INPUT$WAITING '0FFH']
[MACRO IN '[READ]']
[MACRO OUT '[WRITE]']

/**************** MODULE DECLARATIONS ****************/

DECLARE (MONITOR, BOOTSTRAP, RESUME$EXECUTION) LABEL;
DECLARE CONTINUE LABEL;
DECLARE DIR DATA (0);
DECLARE SAVHL DATA (0,0);
DECLARE I DATA (0);
DECLARE TN DATA (0);
DECLARE RST$FLAG DATA (0);

/********************* PROCEDURES *********************/

BUMP$TASK:  PROCEDURE;
/*******************************************************/
/* THIS PROCEDURE DELETES A TASK FROM THE SYSTEM WHEN */
/* AN IRRECOVERABLE MINI-DISK ERROR OCCURS.           */
/* CALLED BY: SWAP, BOOTSTRAP                         */
/*******************************************************/
    C=[BUMP]; CALL [MTS];
    E=[HARDWARE$ERROR]; CALL [MTS$MSG];
    END BUMP$TASK;

SWAP:  PROCEDURE;
/*******************************************************/
/* THIS PROCEDURE SWAPS A TERMINAL TASK BETWEEN MEMORY*/
/* AND THE APPROPRIATE MINI-DISK SWAP FILE. THE DIR-  */
/* ECTION OF THE SWAP, I.E. IN OR OUT, IS DETERMINED  */
/* BY THE VALUE OF THE VARIABLE DIR. THE              */
/* PROCEDURE ALSO MODIFIES TCT$STATUS TO REFLECT THE  */
/* CURRENT LOCATION OF THE SWAP IMAGE.                */
```

```
/* INPUT: DIR - DIRECTION OF SWAP                               */
/*               1 = IN                                         */
/*               2 = OUT                                        */
/* CALLED BY: MONITOR                                           */
/**************************************************************/
    /* SET I TO SWAP IMAGE SIZE */
    DE=[TCT$SIZE]; A=[TASK]; CALL [INDEX];
    I=(A=M(HL));
    /* MODIFY TCT$STATUS */
    HL=[TCT$STATUS]+BC;
    M(HL)=(A=M(HL) \\ 0C0H);
    /* SET UP REGISTERS FOR DATA TRANSFER */
    DE=[TCT$BOE]; A=[TASK]; CALL [INDEX2]; CALL [GET];
    DE=[VDC$DRIVE]; /* SWAP BASE */  A=I;
    DO WHILE (A::0) !ZERO;
        L=(A=DIR); CALL [MINI$DISK];
        IF [DISK$ERROR] THEN
            DO;  /* BUMP TASK OFF SYSTEM */
            CALL BUMP$TASK;
            IF (A=DIR; A::[IN]) ZERO GOTO MONITOR;
            RETURN;
            END;
        BC=BC+1; DE=(HL=200H+DE);
        I=(A=I-1);
    END;  /* WHILE */
    END SWAP;

WRITE$REC:  PROCEDURE;
/**************************************************************/
/* THIS PROCEDURE COPIES THE CONTENTS OF THE SYSTEM     */
/* STATE BLOCK TO THE RECOVERY FILE. THE VARIABLE       */
/* REC$FILE MUST BE SET TO THE FILE'S SECTOR ADDRESS    */
/* BEFORE CALLING WRITE$REC.                            */
/* CALLED BY: BOOTSTRAP, RESUME$EXECUTION               */
/**************************************************************/
    HL=[REC$FILE];
    DE=3C7EH; /* ADDRESS OF THE BASE OF SSB */
    C=(A=M(HL));  HL=HL+1;  B=(A=M(HL));
    L=[WRITE]; CALL [MINI$DISK];
    IF [DISK$ERROR] THEN
        DO; DISABLE; HALT; END;
    BC=BC+1; DE=(HL=200H+DE);
    L=[WRITE]; CALL [MINI$DISK];
    IF [DISK$ERROR] THEN
        DO; DISABLE; HALT; END;
    END WRITE$REC;

MONITOR:
/**************************************************************/
/* THIS ROUTINE IS THE TASK MANAGER OR SCHEDULER        */
/* WHICH CONTROLS THE ALLOCATION OF THE CPU TO COM-     */
/* PETING TERMINAL TASKS.  IT PERFORMS THIS FUNCTION    */
/* BY SEQUENTIALLY SCANNING THE TCT$STATUS BYTE         */
/* ASSOCIATED WITH EACH TERMINAL LOOKING FOR A TASK     */
/* REQUIRING THE CPU.  THE EFFECT PRODUCED IS THAT      */
/* OF A ROUND-ROBIN SCHEDULING ALGORITHM.  WHILE THE    */
/* MONITOR IS LOOPING, IT INITIATES SWAPPING AS         */
/* REQUIRED.                                            */
/* CALLED BY: MTS$IPL, TERM$BLOCK (SVC MOD), SWAP,      */
/* TIMER$HDLR (INT MOD), BOOTSTRAP, QUIT (SVC MOD)      */
/**************************************************************/
    SP=[SYS$STACK$TOP];  /* SET STACK POINTER */
    /* LOCK OUT SWAPPING */
    [LOCK]=(A=[LOCK] \ 01H);
    /* INITIALIZE RESTART FLAG */
    RST$FLAG=(A=0);
    /* INITIALIZE TEMP TASK COUNTER */
    TN=(A=[TASK]);
    LOOP:  /* SEARCH FOR READY TASK */
        TN=(A=TN+1, 803H); /* INCREMENT TASK NUMBER */
        /* TEST FOR INACTIVE TASK */
        DE=[TCT$STATUS]; CALL [INDEX]; A=M(HL);
        IF (A::0) ZERO GOTO LOOP;
```

```
        /* TEST BIT 0 - BOOTSTRAP LOAD */
        IF (A=>A) CY THEN
            DO;
            DE=[TCT$STATUS]; A=[TASK]; CALL [INDEX];
            IF (A=<M(HL)) CY
              & (B=(A=TN); A=[TASK]-B) !ZERO THEN
                DO; DIR=(A=[OUT]); CALL SWAP; END;
            [TASK]=(A=TN);
            GOTO BOOTSTRAP;
            END;
        /* TEST BIT 1 - MCP */
        IF (A=>A) CY THEN
            DO;
            A=[TASK]; STACK=PSW;  /* SAVE OLD TASK NR */
            [TASK]=(A=TN); CALL [MCP];
            DE=[TCT$STATUS]; A=TN; CALL [INDEX];
            M(HL)=(A=M(HL)  & 0FDH);
                            /* RESET BIT 1 */
            PSW=STACK; [TASK]=A;
                            /* RESTORE OLD TASK NR */
            TN=(A=TN-1, &03H); GOTO LOOP;
                            /* CONTINUE WITH TASK AFTER */
                            /* SYSTEM CALL PROCESSED    */
            END;
        /* SKIP BIT 2 - NOT USED */
        A=>A;
        /* SKIP BIT 3 - NOT USED */
        A=>A;
        /* TEST BIT 4 - RST 7 */
        IF (A=>A) CY THEN
            DO;
            STACK=PSW;
            RST$FLAG=(A=0FFH);
            /* RESET TCT$STATUS BITS 4 AND 5 */
            DE=[TCT$STATUS]; A=TN; CALL [INDEX];
            M(HL)=(A=M(HL) & 0CFH);
            PSW=STACK;
            A=>A; /* ADJUST ACCUMULATOR FOR NEXT TEST */
            GOTO CONTINUE;
            END;
        /* TEST BIT 5 - BLOCKED FOR TERMINAL I/O */
        IF (A=>A) CY THEN
            DO;
            A=TN; CALL [GET$TERM$STATUS];
            IF (A::[INPUT$WAITING]) ZERO THEN
                DO; /* NO LONGER BLOCKED */
                IF (B=(A=TN); A=[TASK]-B) !ZERO THEN
                    DO; /* TASK, TN NOT EQUAL */
                    DE=[TCT$STATUS]; A=[TASK];
                    CALL [INDEX];
                    IF (A=<M(HL)) CY THEN
                        DO; /* SWAP OUT OLD IMAGE */
                        DIR=(A=[OUT]);
                        CALL SWAP;
                        END;
                    /* SWAP IN NEW IMAGE */
                    [TASK]=(A=TN);
                    DIR=(A=[IN]); CALL SWAP;
                    END;
                CALL [READ$TERMINAL];
                [SWAP$STACK1]=A;
                /* RESET TCT$STATUS BIT 5 */
                DE=[TCT$STATUS]; A=[TASK]; CALL [INDEX];
                M(HL)=(A=M(HL)  & 0DFH);
                GOTO RESUME$EXECUTION;
                END
            ELSE GOTO LOOP;
            END;
CONTINUE:
        /* TEST BIT 6 - RESUME EXECUTION - FM DISK */
        IF (A=>A) CY THEN
            DO;
            DE=[TCT$STATUS]; A=[TASK]; CALL [INDEX];
```

```
            IF (A=<M(HL)) CY THEN
                DO; DIR=(A=[OUT]); CALL SWAP; END;
            [TASK]=(A=TN);
            DIR=(A=[IN]); CALL SWAP;
            GOTO RESUME$EXECUTION;
            END;
        /* BIT 7 SET - RESUME EXECUTION - IN MEMORY */
        GOTO RESUME$EXECUTION;
        /* END MONITOR */

BOOTSTRAP:
/*********************************************************/
/* THIS ROUTINE EXAMINES THE DISK MAP FOR THE CURRENT */
/* TASK; DETERMINES THE VIRTUAL DISK ATTACHED TO DRIVE*/
/* A; LOADS THE FIRST 512 BYTES FROM THE DISK INTO    */
/* MEMORY STARTING AT THE BASE OF THE USER SWAP AREA; */
/* AND THEN TRANSFERS CONTROL TO THE CODE JUST LOADED.*/
/*********************************************************/
    /* DETERMINE DISK NR ATTACHED TO DRIVE A */
    DE=[TCT$DM]; A=[TASK]; CALL [INDEX8];
    A=M(HL)  8 1FH;
    /* DETERMINE BOE FOR DISK */
    DE=[DMT$BOE]; CALL [INDEX2]; CALL [GET];
    /* READ FIRST SECTOR ON VIRTUAL DISK */
    DE=[MEM$BASE]; L=[READ]; CALL [MINI$DISK];
    IF [DISK$ERROR] THEN
        DO;  /* BUMP TASK OFF SYSTEM */
        CALL BUMP$TASK;
        CALL WRITE$REC;
        GOTO MONITOR;
        END;
    /* UPDATE SYSTEM STATUS */
    DE=[TCT$STATUS]; A=[TASK]; CALL [INDEX];
    A=M(HL) \ 80H;  /* SET BIT 7 */
    M(HL)=(A=A  8 0FEH);  /* RESET BIT 0 */
    CALL WRITE$REC;
    [TASK$TIMER]=(A=[TIMESLICE]);  /* RESET TASK$TIMER */
    [LOCK]=(A=[LOCK]  8 0FEH);  /* UNLOCK SWAPPING */
    SP=0FEFFH;
    GOTO [MEM$BASE];
    /* END BOOTSTRAP */

RESUME$EXECUTION:
/*********************************************************/
/* THIS ROUTINE TRANSFERS CONTROL BACK TO A USER TASK */
/* WHICH HAS BEEN SWAPPED INTO MEMORY.                */
/*********************************************************/
    CALL WRITE$REC;
    /* UPDATE SYSTEM STATUS */
    [TASK$TIMER]=(A=[TIMESLICE]);  /* RESET TASK$TIMER */
    /* RESTORE ORIGINAL VALUE OF STACK POINTER */
    /* AND ALL REGISTERS FROM SWAP$STACK       */
    SP=[SWAP$STACK];
    PSW=STACK; BC=STACK; DE=STACK; HL=STACK;
    SAVHL=HL; HL=STACK;
    SP=HL;  /* RESTORE USER SP */
    STACK=PSW; DISABLE;
    [LOCK]=(A=[LOCK]  8 0FEH);  /* UNLOCK SWAPPING */
    HL=SAVHL; /* RESTORE USERS HL */
    IF (A=RST$FLAG; A::0FFH) ZERO THEN
        DO;  /* RST 7 REQUESTED */
        PSW=STACK; ENABLE;
        CALL 38H; /* RST 7 */
        RETURN;
        END
    ELSE
        DO;  /* NORMAL RETURN */
        PSW=STACK; ENABLE;
        RETURN; /* RETURNS TO INTERRUPT POINT */
                /* IN USER SWAP IMAGE         */
        END;
    /* END RESUME$EXECUTION */
```

```
EOF
/*************************************************************/
/****************** UTILITY PROCEDURES *****************/
/*************************************************************/

/*************** INTERNAL MACROS **********************/
[MACRO ABNORMAL$COMPLETION '0FFH']
[MACRO HARDWARE$ERROR '8']
[MACRO MTS$MSG '1837H']
[MACRO READ '1']
[MACRO PRT$SEC '3FE2H']

INDEX:  PROCEDURE;
/*************************************************************/
/* GIVEN THE BASE ADDRESS OF A BYTE VECTOR AND AN         */
/* INDEX VALUE, THIS PROCEDURE CALCULATES THE ADDRESS */
/* OF THE INDEXED ENTRY.                                   */
/* INPUT:   A - INDEX VALUE (USUALLY TASK)                 */
/*         DE - BASE ADDRESS OF VECTOR                     */
/* OUTPUT: BC - INDEX VALUE                                */
/*         DE - SAME AS INPUT                              */
/*         HL - CALCULATED ADDRESS OF INDEXED ENTRY        */
/* CALLED BY: SWAP, MONITOR, VAL$DISK, CLEAR$FLAG,         */
/*            ATTACH, LOGIN, QUIT, SIZE, TERM$BLOCK,       */
/*            SELECT$DRIVE, RECOVER$STATUS$LINE,           */
/*            PROTECT, RESTRICT, UNPROTECT                 */
/*************************************************************/
   B=0; C=A;
   HL=BC+DE;
   END INDEX;

INDEX2:  PROCEDURE;
/*************************************************************/
/* GIVEN THE BASE ADDRESS OF AN ADDRESS VECTOR AND AN */
/* INDEX VALUE, THIS PROCEDURE CALCULATES THE ADDRESS */
/* OF THE LOW ORDER BYTE OF THE INDEXED ENTRY.             */
/* INPUT:   A - INDEX VALUE (USUALLY TASK)                 */
/*         DE - BASE ADDRESS OF VECTOR                     */
/* OUTPUT: BC - CALCULATED OFFSET = 2 * INDEX VALUE        */
/*         DE - SAME AS INPUT                              */
/*         HL - CACULATED ADDRESS OF INDEXED ENTRY         */
/* CALLED BY: SWAP, BOOTSTRAP, SIZE, SELECT$DRIVE          */
/*************************************************************/
   B=0; C=(A=<<A);
   HL=BC+DE;
   END INDEX2;

INDEX4:  PROCEDURE;
/*************************************************************/
/* INPUT:   A - INDEX VALUE                                */
/*         DE - BASE ADDRESS OF VECTOR                     */
/* OUTPUT: BC - CALCULATED OFFSET = 4 * INDEX VALUE        */
/*         DE - SAME AS INPUT                              */
/*         HL - CALCULATED ADDRESS OF INDEXED ENTRY        */
/* CALLED BY: VAL$KEY, UPDATE$CNFG$FILE, PROTECT,          */
/*            RESRTICT, UNPROTECT                          */
/*************************************************************/
   B=0; C=(A=<<(A=<<A));
   HL=BC+DE;
   END INDEX4;

INDEX8:  PROCEDURE;
/*************************************************************/
/* INPUT:   A - INDEX VALUE                                */
/*         DE - BASE ADRESS OF VECTOR                      */
/* OUTPUT: BC - CALCULATED OFFSET = 8 * INDEX VALUE        */
/*         DE - SAME AS INPUT                              */
/*         HL - CALCULATED ADDRESS OF INDEXED ENTRY        */
/* CALLED BY: BOOTSTRAP, VAL$DRIVE, CLEAR$DM, ATTACH, */
/*            SELECT$DRIVE, RECOVER$STATUS$LINE,           */
/*            UPDATE$CNFG$FILE                             */
/*************************************************************/
```

```
    B=0; C=(A=<<(A=<<(A=<<A)));
    HL=BC+DE;
    END INDEX8;

PUT: PROCEDURE;
/***************************************************************/
/* STORE A TWO BYTE ADDRESS IN A SPECIFIED VECTOR.             */
/* INPUT: DE - ADDRESS TO BE STORED                            */
/*        HL - BASE ADDRESS OF VECTOR                          */
/* OUTPUT: BC - UNCHANGED                                      */
/*         DE - SAME AS INPUT                                  */
/*         HL - BASE ADDRESS + 1                               */
/* CALLED BY: RECOVER, INITIALIZE                              */
/***************************************************************/
    M(HL)=E; HL=HL+1; M(HL)=D;
    END PUT;

GET: PROCEDURE;
/***************************************************************/
/* FETCH A TWO BYTE ADDRESS FROM A SPECIFIED VECTOR.           */
/* INPUT: HL - BASE ADDRESS OF VECTOR                          */
/* OUTPUT: BC - ADDRESS FETCHED                                */
/*         DE - ADDRESS FETCHED                                */
/*         HL - BASE ADDRESS + 1                               */
/* CALLED BY: SWAP, BOOTSTRAP, SIZE, SELECT$DRIVE              */
/***************************************************************/
    E=M(HL); HL=HL+1; D=M(HL); BC=DE;
    END GET;

MOVBUF: PROCEDURE;
/***************************************************************/
/* THIS IS A GENERAL PURPOSE UTILITY PROCEDURE WHICH           */
/* MOVES A SPECIFIED NUMBER OF BYTES FROM A SOURCE             */
/* BUFFER TO A DESTINATION BUFFER.                             */
/* INPUT: BC - NUMBER OF BYTES TO BE MOVED                     */
/*        DE - BASE ADDRESS OF SOURCE BUFFER                   */
/*        HL - BASE ADDRESS OF DESTINATION BUFFER              */
/* CALLED BY: MTS$IPL, ABORT$IPL, SEARCH$DIRECTORY,            */
/*            LOGIN, READ$FLOPPY, WRITE$FLOPPY,                */
/*            INITIALIZE, TIMER$HDLR (INT MOD)                 */
/***************************************************************/
    REPEAT;
        M(HL)=(A=M(DE));
        HL=HL+1; DE=DE+1;
    UNTIL (BC=BC-1; A=0; A::C) ZERO & (A::B) ZERO;
    END MOVBUF;

MINI$DISK: PROCEDURE;
/***************************************************************/
/* THIS PROCEDURE TRANSFERS A SPECIFIED 512 BYTE               */
/* BUFFER BETWEEN MEMORY AND THE MINI-DISK. THE DIR-           */
/* ECTION OF THE TRANSFER IS INDICATED BY THE OP CODE.*/
/* ERROR PROCESSING IS LIMITED TO CHECKING THE STATUS */
/* RETURNED BY THE MINI-DISK CONTROLLER FOR 'NORMAL            */
/* COMPLETION.' WHEN ANY OTHER STATUS IS RETURNED THE */
/* ROUTINE DISPLAYS THE MESSAGE 'HARDWARE ERROR' ON            */
/* THE TERMINAL CURRENTLY ALLOCATED THE CPU AND                */
/* ABORTS THE OPERATION.                                       */
/* INPUT: BC - MINI-DISK SECTOR NUMBER                         */
/*        DE - DMA BUFFER BASE ADDRESS                         */
/*         L - OPERATION CODE:                                 */
/*                1 = READ                                     */
/*                2 = WRITE                                    */
/*                3 = WRITE/VERIFY                             */
/* OUTPUT: A - RETURNS 00H IF COMPLETION NORMAL,               */
/*             OTHERWISE RETURNS FFH                           */
/*        BC - SAME AS INPUT                                   */
/*        DE - SAME AS INPUT                                   */
/* CALLED BY: READ$DIRECTORY, RECOVER, INITIALIZE,             */
/*            WRITE$REC,SWAP,BOOTSTRAP, READ$BUF,              */
/*            WRITE$BUF, READ$PRT$BUF, WRITE$PRT$BUF           */
/***************************************************************/
    /* SET DCB CHECKSUM IN LOCN 4CH */
```

```
    M(4CH)=(A=0\\B,\\C,\\D,\\E,\\L);
    /* SET SECTOR NUMBER IN DISK CONTROL BLOCK */
    M(45H)=(A=B); M(44H)=(A=C);
    /* DEBUG - TO ENSURE SYCOR SYSTEM IS NEVER OVERWRITTEN */
    STACK=HL; HL=0B7H;
    L=(A=L-C); H=(A=H-D);
    IF !CY THEN CALL 10H;
    HL=STACK;
    /* SET DMA BUFFER ADDRESS IN DISK CONTROL BLOCK */
    M(43H)=(A=D); M(42H)=(A=E);
    /* INITIATE OPERATION */
    M(40H)=(A=L);
    /* WAIT UNTIL OPERATION COMPLETE */
    REPEAT;
        A=M(41H);
    UNTIL (A::0) !ZERO;
    /* TEST A FOR COMPLETION STATUS */
    IF (M(41H)=(A=A-1)) ZERO RETURN;  /* NORMAL COMPLETION */
    M(41H)=(A=0);
    E=[HARDWARE$ERROR];
    CALL [MTS$MSG];
    A=[ABNORMAL$COMPLETION];
    END MINI$DISK;

READ$PRT$BUF:  PROCEDURE;
/*******************************************************/
/* THIS ROUTINE IS CALLED INITIALLY BY WRITE$PRINTER */
/* IN ORDER TO FILL THE PRINT BUFFER WITH THE FIRST  */
/* SECTOR OF THE MTSPRT FILE PRIOR TO OUTPUT OF THE  */
/* FIRST CHARACTER TO THE PRINTER.  SUBSEQUENT CALLS */
/* ARE MADE BY THE PRINTER$HDLR IN THE INTERRUPT$    */
/* CONTROLLER UNTIL THE END OF FILE IS ENCOUNTERED   */
/* CALLED BY: WRITE$PRINTER, PRINTER$HDLR, RECOVER   */
/*******************************************************/
    HL=[PRT$SEC];   C=M(HL);   HL=HL+1;
    B=M(HL);   /* BC HAS CURRENT MTSPRT SECTOR # */
    DE=100H;   L=[READ];
    CALL MINI$DISK;
    BC=BC+1;   HL=[PRT$SEC];
    M(HL)=C;   HL=HL+1;   M(HL)=B;
    END READ$PRT$BUF;

EOF
/*********************************************************/
/**************** INITIAL PROGRAM LOAD ****************/
/*********************************************************/


/************* INTERMODULE LINKAGE MACROS *************/

[INT MB M2B SB TB]
[MB:=0300H] [M2B:=0600H] [SB:=1F00H] [TB:=1000H]
[MACRO MONITOR '[HEX MB + 0AAH]']
[MACRO MOVBUF '[HEX M2B + 41H]']
[MACRO PUT '[HEX M2B + 31H]']
[MACRO MINI$DISK '[HEX M2B + 54H]']
[MACRO INDEX '[HEX M2B + 3H]']
[MACRO INDEX8 '[HEX M2B + 24H]']
[MACRO READ$PRT$BUF '[HEX M2B + 9EH]']
[MACRO MTS '[HEX SB + 0]']
[MACRO MTS$MSG '[HEX TB + 837H]']
[MACRO CLEAR$STATUS$LINE '[HEX TB + 827H]']
[MACRO TERMINAL$STATUS '[HEX TB + 8D2H]']
[MACRO READ$TERMINAL '[HEX TB + 8DCH]']
[MACRO WRITE$TERMINAL '[HEX TB + 93CH]']
[MACRO SIZE$MSG '[HEX TB + 864H]']
[MACRO STATUS$MSG '[HEX TB + 88CH]']
[MACRO BUF$PTR '3C40H']
[MACRO TASK '3E80H']
[MACRO REC$FILE '3E81H']
[MACRO TCT$STATUS '3E85H']
[MACRO TCT$SIZE '3EA9H']
[MACRO TCT$EOE '3EB5H']
```

```
[MACRO TCT$DM '3E89H']
[MACRO CNFG$FILE '3E83H']
[MACRO DMT$FLAG '3EBDH']
[MACRO DMT$BOE '3EDDH']
[MACRO DMT$EOE '3F1DH']
[MACRO DMT$KEY '3F5DH']
[MACRO PRT$CNTRL '3FDDH']
[MACRO PRT$BOE '3FDEH']
[MACRO PRT$EOE '3FE0H']
[MACRO PRT$SEC '3FE2H']
[MACRO SYS$STACK '3C6AH']
/* .MDBUF = 3C7EH - USED IN RECOVER */

/*************** GENERAL PURPOSE MACROS ***************/

[INT IPL$OFFSET]
[IPL$OFFSET:=55C7H]  /* ADDR OF LABEL SHIFT + 1200H */

/**************** MODULE DECLARATIONS *****************/

DECLARE (I,J) BYTE;
DECLARE MAX(2) BYTE;
    /* ADDRESS OF LAST ENTRY + 1 IN DIRECTORY IMAGE */
DECLARE REC$NAME(9) BYTE INITIAL ('.MTSRCVR$');

/******************* PROCEDURES **********************/

ABORT$IPL:  PROCEDURE(MSG);
/*****************************************************/
/* WHENEVER A CONDITION OCCURS DURING THE IPL PROCESS */
/* WHICH PREVENTS NORMAL COMPLETION OF THE IPL THIS   */
/* PROCEDURE IS CALLED TO TERMINATE EXECUTION AND     */
/* DISPLAY AN ERROR MESSAGE AT TERMINAL 0.            */
/* INPUT: MSG - BASE ADDRESS OF ERROR MESSAGE         */
/*             TERMINATED BY '$'                      */
/* CALLED BY: READ$DIRECTORY, INITIALIZE, RECOVER,    */
/*****************************************************/
    DECLARE ABORT$MSG DATA ('IPL ABORTED - ');
    /* DISPLAY 'IPL ABORTED' AT TERMINAL 0 */
    BC=14; DE=.ABORT$MSG; HL=0700H;
    CALL [MOVBUF];
    HL=MSG; A='$';
    DO C=0 BY C=C+1 WHILE (A::M(HL)) !ZERO;
        HL=HL+1;  /* COUNT CHARS IN MSG */
    END;
    B=0; DE=(HL=MSG); HL=070EH;
    CALL [MOVBUF];  /* DISPLAY MSG AT TERMINAL 0 */
    DISABLE; HALT;
    END ABORT$IPL;

READ$DIRECTORY:  PROCEDURE;
/*****************************************************/
/* DURING THE INITIALIZATION PROCESS IT IS NECESSARY  */
/* TO DETERMINE THE SECTOR NUMBERS  OF SEVERAL SYSTEM */
/* FILES WHICH RESIDE ON THE MINI-DISK IN SYCOR FORMAT.*/
/* MULTIPLE DIRECTORY SEARCHES COULD LEAD TO REPEATEDLY*/
/* READING THE SAME BLOCK OF MINI-DISK SECTORS. TO    */
/* ELIMINATE MOST OF THESE UNNECESSARY READ OPERATIONS */
/* THIS PROCEDURE READS THE ENTIRE SYCOR DIRECTORY    */
/* INTO MEMORY AT ONE TIME, THUS REDUCING THE OVERHEAD */
/* INVOLVED IN MULTIPLE SEARCHES.                     */
/* CALLED BY: MTS$IPL                                 */
/*****************************************************/
    [INT SYCOR$DIR$BASE] /* SYCOR DIRECTORY BASE */
    [SYCOR$DIR$BASE:=20H] /* SECTOR NUMBER        */
    DECLARE READ$MSG(22) BYTE INITIAL ('CANNOT READ DIRECTORY$');
    /* SET UP REGISTERS FOR DISK READ */
    BC=[HEX SYCOR$DIR$BASE];
    DE=5200H;  /* 5000H + 200H = DIRECTORY BASE ADDRESS */
    /* READ NUMBER OF SECTORS INDICATED */
    /* IN DIRECTORY HEADER RECORD       */
    REPEAT;
        L=1; /* READ */ CALL [MINI$DISK];
```

```
        IF (A::0) !ZERO    /* DISKERROR */
        CALL ABORT$IPL(.READ$MSG);
        DE=(HL=200H+DE); BC=BC+1;
    UNTIL (A=M(520AH); /* DIR BASE + 0AH */ A::C) CY;
    /* CALCULATE ADDRESS OF LAST ENTRY + 1 IN IMAGE */
    B=[HEX SYCOR$DIR$BASE-1];
    A=A-B;  /* A = NR SECTORS IN DIRECTORY */
    D=(A=<<A);
    E=0;  /* DE = NR SECTORS * 512 */
    HL=5201H; /* DIR BASE + 1 */ HL=HL+DE;
    MAX=HL;
    END READ$DIRECTORY;

SEARCH$DIRECTORY:  PROCEDURE;
/*********************************************************/
/* GIVEN THE BASE ADDRESS OF A VECTOR CONTAINING THE    */
/* NAME OF A FILE IN SYCOR FORMAT, THIS ROUTINE WILL    */
/* SEARCH THE DIRECTORY IMAGE READ INTO MEMORY BY       */
/* READ$DIRECTORY. IF THE FILE ENTRY IS FOUND, THE      */
/* BOE AND EOE VALUES ARE RETURNED.                     */
/* INPUT: DE - BASE ADDRESS OF FILENAME VECTOR          */
/*              ASSUMED TO BE 8 BYTES LONG              */
/* OUTPUT: BC - BOE OR FFFFH IF FILE NOT FOUND          */
/*         DE - EOE OR FFFFH IF FILE NOT FOUND          */
/* CALLED BY: INITIALIZE, RECOVER                       */
/*********************************************************/
    DECLARE LOOP LABEL;
    /* MOVE FILENAME TO LAST ENTRY + 1 */
    BC=8; HL=MAX; CALL [MOVBUF];
    DE=5241H;  /* DIR BASE + 41H - ADDRESS OF FIRST ENTRY */
    LOOP:  /* ADVANCE TO NEXT ENTRY */
        STACK=DE; HL=MAX; B=8;
        REPEAT;  /* COMPARE CHAR BY CHAR */
            IF (A=M(DE); A::M(HL)) ZERO
             \ (IF (A::0) ZERO  & (A=M(HL)-20H) ZERO
               THEN CY=1 ELSE CY=0) CY
            THEN  /* CHAR MATCH */
               DO;
               DE=DE+1; HL=HL+1;
               END
            ELSE  /* NON-MATCH */
               DO; DE=STACK;
               DE=(HL=40H+DE);
               GOTO LOOP;
               END;
        UNTIL (B=B-1) ZERO;
    SP=(HL=2+SP); /* CLEAR STACK */
    /* FALLING THRU LOOP MEANS NAMES MATCH,         */
    /* MUST TEST FOR SUCCESS OR FAILURE OF SEARCH */
    IF (A=MAX(1); A::D) CY  /* X::Y=CY => X<Y */
    \(IF ZERO  & (A=MAX(0); A::E) CY THEN CY=1 ELSE CY=0)
        CY THEN
        DO;  /* SEARCH FAILED */
        BC=0FFFFH; DE=BC;
        END
    ELSE DO;  /* SEARCH SUCCESSFUL */
        HL=3+DE;
        C=M(HL); HL=HL+1;
        B=M(HL); HL=HL+1;
        E=M(HL); HL=HL+1;
        D=M(HL);
        END;
    END SEARCH$DIRECTORY;

RECOVER$STATUS$LINE:  PROCEDURE;
/*********************************************************/
/* ROUTINE TO BUILD AND DISPLAY THE STATUS LINE WHEN    */
/* A RECOVERY IS PERFORMED. THIS ROUTINE DISPLAYS ALL   */
/* ACTIVE DRIVES AND ASSOCIATED DISKS AND DISPLAYS      */
/* SWAP IMAGE SIZE FOR EACH TASK.                       */
/* CALLED BY: RECOVER                                   */
/*********************************************************/
    DECLARE SAVE$TASK BYTE;
```

```
    HL=[TASK];
    SAVE$TASK=(A=M(HL)); /* SAVE CURRENT TASK NUMBER */
    I=(A=0);  /* INITIALIZE TASK COUNTER */
    /* CHECK TCT$STATUS FOR ACTIVE TASK  */
    REPEAT; /* FOR ALL 4 TASKS */
         DE=[TCT$STATUS]; CALL [INDEX];
         IF (A=M(HL); A::0) !ZERO THEN  /* TASK ACTIVE */
           DO;
              HL=[TASK]; M(HL)=(A=I); /* SET TASK NUMBER */
              DE=[TCT$DM]; CALL [INDEX8];
              J=(A=0);  /* INITIALIZE DRIVE COUNTER */
              REPEAT; /* FOR ALL 8 DRIVES */
                   IF (A=<M(HL)) CY THEN
                     DO;  /* DRIVE IN USE */
                       B=(A=J); C=(A=M(HL) & 1FH);
                       IF (A=M(HL) & 40H) !ZERO
                            THEN A=72H ELSE A=' ';
                       STACK=HL;
                       /* DISPLAY ACTIVE DRIVE AND DISK */
                       CALL [STATUS$MSG];
                       HL=STACK;
                     END;
              UNTIL (HL=HL+1; J=(A=J+1); A::8) ZERO;
              /* DISPLAY SIZE MESSAGE */
              DE=[TCT$SIZE]; A=I; CALL [INDEX];
              CY=0; A=>M(HL); CALL [SIZE$MSG];
           END;
    UNTIL (I=(A=I+1); A::4) ZERO;
    /* RESTORE CURRENT TASK NUMBER */
    HL=[TASK]; M(HL)=(A=SAVE$TASK);
    END RECOVER$STATUS$LINE;

RECOVER:  PROCEDURE;
/*********************************************************/
/* MTS HAS BEEN DESIGNED SO THAT THE SYSTEM STATE AT    */
/* ANY INSTANT IS DEFINED BY A COMPACT, CONTIGUOUS      */
/* GROUP OF BYTES KNOWN AS THE SYSTEM STATE BLOCK.      */
/* EACH TIME THAT SWAPPING OCCURS THE SSB IS WRITTEN    */
/* TO THE MINI-DISK FILE .MTSRCVR. IF THE TASK JUST     */
/* SWAPPED IN CAUSES A SYSTEM CRASH, RECOVERY IS        */
/* ACCOMPLISHED BY REBOOTING MTS AND ANSWERING 'Y' TO  */
/* THE RECOVERY QUERY. MTS WILL READ .MTSRCVR BACK      */
/* INTO THE SSB, DELETE THE OFFENDING TASK, AND         */
/* CONTINUE WITH THE NEXT READY TASK.                   */
/* NOTE: THIS PROCEDURE USES THE FACT THAT THE BOE      */
/*       AND EOE VALUES RETURNED BY SEARCH$DIRECTORY    */
/*       ARE EQUAL FOR A SINGLE-SECTOR FILE.            */
/* CALLED BY: MTS$IPL                                   */
/*********************************************************/
    /* FIND MINI-DISK SECTOR ADDRESS OF .MTSRCVR */
    DE=.REC$NAME;
    CALL SEARCH$DIRECTORY;
    IF (A=B; A::0FFH) ZERO CALL ABORT$IPL(.REC$NAME);
    /* READ .MTSRCVR INTO SSB */
    DE=BC;  /* MOVE BOE TO DE */
    HL=[REC$FILE]; CALL [PUT];
    DE=3C7EH;  /* ADDRESS OF SSB - .MDBUF */
    DISABLE;
    I=(A=0);
    REPEAT;
    L=1; /* READ */ CALL [MINI$DISK];
    IF (A::0) !ZERO  /* DISK ERROR */
    CALL ABORT$IPL(.REC$NAME);
    BC=BC+1; DE=(HL=200H+DE);
    UNTIL (I=(A=I+1); A::2) ZERO;
    IF (A=M([PRT$CNTRL]); A=<A; A=<A) CY THEN
    DO; /* COMPLETE PRINTING TASK */
        HL=[PRT$SEC];  DE=[PRT$BOE];
        M(HL)=(A=M(DE));  HL=HL+1;  DE=DE+1;
        M(HL)=(A=M(DE));
        /*  SET BUFFER PTR TO 100H  */
        HL=[BUF$PTR];  M(HL)=0; HL=HL+1;
        M(HL)=1;
```

```
        CALL [READ$PRT$BUF];
        OUT(8AH)=(A=0);
    END;
    ENABLE;
    /* DELETE TASK CAUSING CRASH */
    C=-2;  /* BUMP */
    CALL [MTS];
    E=9;  /* TASK DELETED */
    CALL [MTS$MSG];
    CALL RECOVER$STATUS$LINE;
    END RECOVER;

INITIALIZE:  PROCEDURE;
/*******************************************************/
/* THE SYSTEM STATE BLOCK CONSISTS OF THREE SETS OF    */
/* VARIABLES: SYSTEM CONTROL, TASK CONTROL TABLE, AND */
/* THE DISK MAP TABLE. THE OBJECT MODULE GENERATED BY */
/* THE ML80 COMPILER CONTAINS INITIAL VALUES FOR       */
/* SYSTEM CONTROL VARIABLES AND MOST OF THE TCT. IN    */
/* ORDER TO INITIALIZE THE REST OF THE TCT IT IS       */
/* NECESSARY TO SEARCH THE SYCOR DIRECTORY IMAGE       */
/* COPIED INTO MEMORY BY READ$DIRECTORY FOR BOE AND    */
/* EOE VALUES FOR THE RECOVERY FILE AND ALL FOUR SWAP */
/* FILES. TO INITIALIZE THE DMT THE VIRTUAL DISK       */
/* CONFIGURATION FILE, .MTSCNFG, MUST BE READ INTO     */
/* MEMORY AND BOE AND EOE VALUES FOR THE VIRTUAL DISK */
/* FILES EXTRACTED FROM THE SYCOR DIRECTORY IMAGE.     */
/* THE PROTECTION ATTRIBUTES FOR EACH VIRTUAL DISK     */
/* ARE ALSO COPIED INTO THE DMT FROM .MTSCNFG.         */
/* NOTE: THIS PROCEDURE USES THE FACT THAT THE BOE     */
/*       AND EOE VALUES RETURNED BY SEARCH$DIRECTORY   */
/*       ARE EQUAL FOR A SINGLE-SECTOR FILE.           */
/* CALLED BY: MTS$IPL                                  */
/*******************************************************/
    DECLARE ENTRY$BASE(2) BYTE;
    DECLARE CNFG$NAME(9) BYTE INITIAL ('.MTSCNFG$');
    DECLARE SWAP$NAME(9) BYTE INITIAL ('.MTSSWP0$');
    DECLARE SYSDISK(9) BYTE INITIAL ('SYS DISK$');
    DECLARE PRT$FILE$NAME(9) BYTE INITIAL ('.MTSPRT $');
    /* SET UP RECOVERY FILE */
    DE=.REC$NAME; CALL SEARCH$DIRECTORY;
    IF (A=B; A::0FFH) ZERO CALL ABORT$IPL(.REC$NAME);
    E=(A=E-C);
    IF (E=E-1) !ZERO THEN CALL ABORT$IPL(.REC$NAME);
    DE=BC; /* SET UP TO STORE BOE */
    HL=[REC$FILE]; CALL [PUT];
    /* SET UP TASK CONTROL BLOCK IN SSB */
    I=(A=4); STACK=(HL=[TCT$EOE]);
    REPEAT;
        DE=.SWAP$NAME; CALL SEARCH$DIRECTORY;
        IF (A=B; A::0FFH) ZERO
            CALL ABORT$IPL(.SWAP$NAME);
        /* CHECK THAT SWAP FILE AT LEAST 16K */
        L=(A=!C,+1); H=(A=!B,++0);
        HL=HL+DE;
        IF (A=L; A::31) CY CALL ABORT$IPL(.SWAP$NAME);
        HL=STACK; CALL [PUT];
        DE=BC; BC=-9;
        HL=HL+BC; CALL [PUT];
        BC=9; STACK=(HL=HL+BC);
        SWAP$NAME(7)=(A=SWAP$NAME(7)+1);
    UNTIL (I=(A=I-1)) ZERO;
    SP=(HL=2+SP);  /* CLEAR STACK */
    DE=.PRT$FILE$NAME; CALL SEARCH$DIRECTORY;
    IF (A=B; A::0FFH) ZERO CALL ABORT$IPL(.PRT$FILE$NAME);
    HL=[PRT$EOE];  CALL [PUT];
    DE=BC; HL=[PRT$BOE]; CALL [PUT];
    /* SET UP DISK MAP TABLE IN SSB */
    DE=.CNFG$NAME; CALL SEARCH$DIRECTORY;
    IF (A=B; A::0FFH) ZERO CALL ABORT$IPL(.CNFG$NAME);
    HL=[CNFG$FILE]; CALL [PUT];
    /* READ CONFIGURATION FILE INTO MEMORY */
    DE=5000H;  /* ADDRESS FOR BASE OF MEMORY */
```

```
      L=1;  /*  READ  */  CALL [MINI$DISK];
      IF (A::0) !ZERO /* DISK ERROR */
      CALL ABORT$IPL(.CNFG$NAME);
      HL=5000H;    /* MEMORY BASE ADDRESS */
      IF (A=M(HL); A::2EH) !ZERO THEN
        DO;  /* FIRST BYTE OF .MTSCNFG NOT '.' */
             /* ELIMINATE SYCOR OVERHEAD FROM .MTSCNFG */
             /* SYCOR OVERHEAD (OH) OF THE FORM: */
             /* 2 BYTES OH, 100H BYTES DATA       */
             /* 4 BYTES OH, 0A0H BYTES DATA       */
          DE=5002H;     /* ADDRESS OF SOURCE */
          BC=100H; /* NUMBER OF BYTES TO BE MOVED */
          CALL [MOVBUF];
          DE=5106H;  /*  ADDRESS OF SOURCE  */
          HL=5100H;    /*  ADDRESS OF DESTINATION  */
          BC=0A0H; /* NUMBER OF BYTES TO BE MOVED */
          CALL [MOVBUF];
          /* UPDATE CONFIGURATION FILE */
          DE=5000H;  HL=[CNFG$FILE];
          C=M(HL); HL=HL+1; B=M(HL);
          L=2; /* WRITE */ CALL [MINI$DISK];
          IF (A::0) !ZERO  /* DISK ERROR */
          CALL ABORT$IPL(.CNFG$NAME);
        END;
      I=(A=0); ENTRY$BASE=(HL=(DE=5000H));
      REPEAT;  /* STEP THRU CONFIGURATION FILE */
          CALL SEARCH$DIRECTORY;
          IF (A=B; A::0FFH) !ZERO THEN
              DO;  /* VIRTUAL DISK (I) EXISTS */
              STACK=BC; B=0; C=(A=<<I);
              HL=[DMT$EOE]+BC; CALL [PUT];
              DE=STACK; HL=[DMT$BOE]+BC; CALL [PUT];
              BC=8; DE=(HL=ENTRY$BASE+BC);
              B=0; C=(A=<<(A=<<I));
              HL=[DMT$KEY]+BC;
              DO B=0 BY B=B+1 WHILE (A=B-4) !ZERO;
                  M(HL)=(A=M(DE));
                  DE=DE+1; HL=HL+1;
              END;
              B=0; C=(A=I);
              HL=[DMT$FLAG]+BC;
              IF (A=M(DE); A::(B='R')) ZERO THEN
                  M(HL)=0DH
              ELSE DO;
                  IF (A::(B='P')) ZERO THEN M(HL)=05H
                  ELSE M(HL)=01H;
                  END;
              END;
          DE=(HL=ENTRY$BASE+(BC=13));
          ENTRY$BASE=HL;
      UNTIL (I=(A=I+1); A::32) ZERO;
      /* CHECK THAT DISK 0 EXISTS */
      HL=[DMT$FLAG];
      IF (A=M(HL)  & 01) ZERO CALL ABORT$IPL(.SYSDISK);
      END INITIALIZE;

MTS$IPL:
/***********************************************************/
/* THIS ROUTINE IS THE INITIAL ENTRY POINT INTO MTS.      */
/* THE SYCOR 440 LOADER TRANSFERS CONTROL HERE AFTER      */
/* THE SYSTEM OBJECT  MODULE HAS BEEN LOADED. IF THE      */
/* SYSTEM LOADER LOADS MTS THE LOAD MODULE IS DIS-        */
/* PLACED BY 1200H SO THIS ROUTINE MOVES MTS TO           */
/* ABSOLUTE LOCATION ZERO THEN TRANSFERS CONTROL TO       */
/* THE MOVED MTS$IPL ROUTINE. DURING                      */
/* IPL ALL PERIPHERAL DEVICES ARE RESET, THEN MTS         */
/* READS THE SYCOR DIRECTORY INTO MEMORY, AND ASKS        */
/* THE OPERATOR AT TERMINAL 0 WHETHER RECOVERY IS         */
/* REQUIRED. IF THE ANSWER IS 'Y' THEN THE PROCEDURE      */
/* RECOVER IS CALLED TO READ THE FILE .MTSRCVR INTO       */
/* THE SYSTEM STATE BLOCK. OTHERWISE THE PROCEDURE        */
/* INITIALIZE IS CALLED TO BUILD AN SSB FROM INFOR-       */
/* MATION CONTAINED IN THE SYCOR DIRECTORY IMAGE AND      */
```

```
/* THE FILE .MTSCNFG. ONCE IPL IS COMPLETE CONTROL IS */
/* TRANSFERRED TO THE PROCESSOR MANAGEMENT SUBMODULE  */
/* WHICH WILL CONTROL ALL SUBSEQUENT PROCESSING.      */
/* CALLED BY: SYCOR SYSTEM LOADER                     */
/*****************************************************/
    DECLARE (WAIT, TEST, CLEAR, SHIFT) LABEL;
    DECLARE IPL$MSG(15) BYTE INITIAL ('RECOVERY? (Y/N)');
    /* SET STACK POINTER */
    DE=20;  /* INDEX TO TOP OF SYS$STACK */
    SP=(HL=[SYS$STACK] + DE);
    /* CLEAR PERIPHERAL INTERRUPTS */
    OUT(2)=(A=1); /* DISABLE TIMER */
    OUT(85H)=(A=10H); /* CLEAR CSST INTERUPT */
    A=IN(8AH); /* CLEAR PRINTER INTERUPT */
    /* IF DEBUGGER IS USED TO LOAD THE SYSTEM*/
    /* SKIP THE MOVE, OTHERWISE RELOCATE MTS */
    /* LOAD$SWITCH EQUALS 0 FOR DISK LOAD    */
    /* AND 0FFH FOR DEBUGGER LOAD            */
    IF (A=0FFH; A::0 /* LOAD$SWITCH */ ) ZERO GOTO CLEAR;
    HL=0000H; DE=1200H;  BC=4600H;  /* MTS SIZE - 4600H */
    SHIFT:
        M(HL)=(A=M(DE));
        HL=HL+1;  DE=DE+1;
    IF (BC=BC-1; A=0; A::C) !ZERO GOTO [HEX IPL$OFFSET];
    IF (A::B) !ZERO GOTO [HEX IPL$OFFSET];
    /* JUMP IS MADE TO SHIFT ADDR + 1200H */
    /* WHILE MTS IS TRANSFERRED          */

    GOTO CLEAR;
    CLEAR:
    /* CLEAR STATUS LINE ON ALL TERMINALS */
    DO I=(A=0) BY I=(A=I+1) WHILE (A=I; A::4) !ZERO;
        CALL [CLEAR$STATUS$LINE];
    END;
    /* READ SYCOR DIRECTORY INTO MEMORY */
    CALL READ$DIRECTORY;
    /* DISPLAY IPL$MSG AT TERMINAL 0 */
    BC=15; DE=.IPL$MSG; HL=0700H;
    CALL [MOVBUF];
    /* ENABLE INTERRUPTS SO TERMINAL MODULE MAY */
    /* BE USED TO PROCESS REPLY TO IPL$MSG      */
    ENABLE;
    OUT(2)=(A=0); /* RESET TIMER */
    /* PROCESS OPERATOR'S REPLY  */
    WAIT: REPEAT;
            CALL [TERMINAL$STATUS];
        UNTIL (A::0) !ZERO;
        CALL [READ$TERMINAL];
        I=A;
        IF (A::0DH) ZERO GOTO TEST;
        REPEAT;
            CALL [READ$TERMINAL];
        UNTIL (A::0DH) ZERO;
    TEST:
        IF (A=I; A::(B='Y')) ZERO
         \ (A::(B=79H)) ZERO
        THEN DO;
                A=0; CALL [CLEAR$STATUS$LINE];
                CALL RECOVER;
             END
        ELSE DO;
             IF (A::(B='N')) !ZERO
               & (A::(B=6EH)) !ZERO
             THEN
                DO; E='?';
                CALL [WRITE$TERMINAL];
                GOTO WAIT;
                END
             ELSE
                DO;
                A=0; CALL [CLEAR$STATUS$LINE];
                CALL INITIALIZE;
                END;
```

```
            END;
    GOTO [MONITOR];
    /* END MTS$IPL */

EOF
```

```
/*******************************************************/
/*                   SERVICE MODULE                    */
/*******************************************************/
/*                                                     */
/* THIS MODULE   PROVIDES THE INTERFACE BETWEEN THE    */
/* USER AND ALL SYSTEM SERVICES.  THESE SERVICES FALL  */
/* GENERALLY INTO TWO CATEGORIES:                      */
/*                                                     */
/*  (1) SYSTEM CALLS - THOSE FUNCTIONS REQUIRED TO     */
/*      ESTABLISH THE DESIRED VIRTUAL MACHINE          */
/*      ENVIRONMENT.                                   */
/*                                                     */
/*  (2) SERVICE CALLS - THOSE FUNCTIONS REQUIRED TO    */
/*      ACCESS THE VIRTUAL DEVICES PROVIDED BY THE     */
/*      VIRTUAL MACHINE ENVIRONMENT.                   */
/*                                                     */
/* SYNTACTICALLY THE TWO TYPES OF PROCEDURE CALL ARE   */
/* IDENTICAL, I.E.                                     */
/*        VALUE = MTS(FID,PARM).                       */
/* EACH CALL TAKES TWO ARGUMENTS, FID IN REGISTER C    */
/* AND PARM IN REGISTERS DE; AND RETURNS A VALUE       */
/* IN THE A REGISTER. THE FORM OF THE ARGUMENTS AND    */
/* THE SIDE EFFECTS ASSOCIATED WITH EACH DIFFERENT     */
/* FUNCTION ARE DISCUSSED IN MORE DETAIL IN THE MTS    */
/* USER'S MANUAL. NOTE HOWEVER THAT THE ARGUMENT REG-  */
/* ISTER ASSIGNMENTS CONFORM TO THE PL/M CONVENTION    */
/* FOR PASSING PARAMETERS. THE FOLLOWING TABLE SUM-    */
/* MARIZES THE ARGUMENT OPTIONS AND CORRESPONDING      */
/* RETURNED VALUES.                                    */
/*                                                     */
/*   FID    NAME             PARM         VALUE        */
/*   ---    ----             ----         -----        */
/*   ------------- SYSTEM CALLS -------------          */
/*    0    ATTACH            LIST         ERROR        */
/*    1    DISPLAY$MSG       ERROR        NONE         */
/*    2    LOGIN             LIST         ERROR        */
/*    3    PROTECT           LIST         ERROR        */
/*    4    QUIT              NONE         NONE         */
/*    5    RESTRICT          LIST         ERROR        */
/*    6    SIZE              SIZE         ERROR        */
/*    7    UNPROTECT         LIST         ERROR        */
/*   ------------ SERVICE CALLS -------------          */
/*    8    TERMINAL$STATUS   NONE         TRUE/FALSE   */
/*    9    READ$TERMINAL     NONE         CHARACTER    */
/*   10    WRITE$TERMINAL    CHARACTER    NONE         */
/*   11    WRITE$PRINTER     CHARACTER    ERROR        */
/*   12    SELECT$DRIVE      DRIVE NR     ERROR        */
/*   13    SET$DMA           DMA ADDRESS  ERROR        */
/*   14    SET$TRACK         TRACK NR     ERROR        */
/*   15    SET$SECTOR        SECTOR NR    ERROR        */
/*   16    READ$FLOPPY       NONE         ERROR        */
/*   17    WRITE$FLOPPY      NONE         ERROR        */
/*                                                     */
/* IN THIS TABLE THE ENTRY 'LIST' UNDER PARM INDICATES*/
/* THAT THE ACTUAL ARGUMENT IS THE ADDRESS OF A LIST   */
/* OF REQUIRED PARAMETERS. WHEN 'NONE' APPEARS UNDER   */
/* THE SAME HEADING, IT MEANS THAT THE PARM ARGUMENT   */
/* IS NOT REQUIRED. IF 'NONE' APPEARS UNDER VALUE THE  */
/* ERROR CODE RETURNED IS ALWAYS ZERO. THE ENTRY       */
/* 'ERROR' INDICATES THAT AN ERROR CODE IS RETURNED.   */
/* THESE CODES ARE DEFINED AS FOLLOWS:                 */
/*                                                     */
/*       CODE              ERROR                       */
/*       ----              -----                       */
/*         0         OPERATION SUCCESSFUL              */
/*         1         INVALID COMMAND                   */
/*         2         DISK NOT AVAILABLE                */
/*         3         DISK IN USE                       */
/*         4         DISK NUMBER ERROR                 */
/*         5         KEY ERROR                         */
/*         6         DRIVE LETTER ERROR                */
/*         7         PRINTER NOT READY                 */
/*         8         HARDWARE ERROR                    */
```

```
/*           9            TASK DELETED                         */
/*          10            DRIVE NOT AVAILABLE                  */
/*          11            PRINTER IN USE                       */
/*          12            OUT OF BOUNDS                        */
/*                                                             */
/* THE SERVICE MODULE IS DIVIDED INTO THREE BASIC              */
/* SUBMODULES:                                                 */
/*                                                             */
/*  (1) USER INTERFACE                                         */
/*      THIS SUBMODULE CONTAINS THE ENTRY POINT INTO           */
/*      THE SERVICE MODULE FROM MCP AND USER PRO-              */
/*      GRAMS. IT IS HERE THAT THE SERVICE REQUEST             */
/*      IS INTERPRETED AND THE APPROPRIATE SERVICE             */
/*      ROUTINE CALLED FOR EXECUTION.                          */
/*                                                             */
/*  (2) SYSTEM CALLS                                           */
/*      THIS SUBMODULE CONTAINS THE SERVICE ROUTINES           */
/*      AND SUPPORTING PROCEDURES WHICH CREATE OR              */
/*      MODIFY THE USER'S VIRTUAL ENVIRONMENT.                 */
/*      THESE ROUTINES ARE INVOKED BY MCP IN RESPONSE          */
/*      TO SYSTEM COMMANDS ENTERED AT A TERMINAL.              */
/*      THEY MAY ALSO BE ACCESSED BY USER PROGRAMS             */
/*      DIRECTLY THROUGH THE USER INTERFACE SUBMODULE.*/
/*                                                             */
/*  (3) SERVICE CALLS                                          */
/*      THIS SUBMODULE CONTAINS SERVICE ROUTINES AND           */
/*      SUPPORTING PROCEDURES WHICH ALLOW A USER               */
/*      PROGRAM TO ACCESS A VIRTUAL TERMINAL OR                */
/*      VIRTUAL FLOPPY DISK.                                   */
/*                                                             */
/*************************************************************/
/*************************************************************/


/*************************************************************/
/******************** USER INTERFACE *******************/
/*************************************************************/


/************** INTERMODULE LINKAGE MACROS ************/

DECLARE GLOB1 COMMON;
[INT MB TB M2B] [M2B:=0600H] [MB:=0300H] [TB:=1000H]
[INT S2B S3B]  [S2B:=2200H]  [S3B:=2700H]
[MACRO MONITOR '[HEX MB +0AAH]']
[MACRO INDEX '[HEX M2B + 3H]']
[MACRO ATTACH '[HEX S2B + 179H]']
[MACRO LOGIN '[HEX S2B + 224H]']
[MACRO PROTECT '[HEX S2B + 2ECH]']
[MACRO QUIT '[HEX S2B + 364H]']
[MACRO BUMP '[HEX S2B + 12BH]']
[MACRO RESTRICT '[HEX S2B + 36EH]']
[MACRO SIZE '[HEX S2B + 3F0H]']
[MACRO UNPROTECT '[HEX S2B + 448H]']
[MACRO WRITE$PRINTER '[HEX S3B + 172H]']
[MACRO SELECT$DRIVE '[HEX S3B + 22CH]']
[MACRO SET$DMA '[HEX S3B + 29EH]']
[MACRO SET$TRACK '[HEX S3B + 2BBH]']
[MACRO SET$SECTOR '[HEX S3B + 2D0H]']
[MACRO READ$FLOPPY '[HEX S3B + 2EAH]']
[MACRO WRITE$FLOPPY '[HEX S3B + 2FEH]']
[MACRO MTS$MSG '[HEX TB + 837H]']
[MACRO TERMINAL$STATUS '[HEX TB + 8D2H]']
[MACRO READ$TERMINAL '[HEX TB + 8DCH]']
[MACRO WRITE$TERMINAL '[HEX TB + 93CH]']
[MACRO PARM0 'GLOB1(10H)']
[MACRO ERROR 'GLOB1(14H)']
[MACRO LOCK 'GLOB1(15H)']
[MACRO SVC$STACK$TOP 'GLOB1(2BH)'] /* SVC$STACK + TOP */
[MACRO TASK 'GLOB1(241H)']
[MACRO TCT$STATUS 'GLOB1(246H)']
[MACRO SWAP$STACK9 'GLOB1(3BFH)']
```

```
/*************** GENERAL PURPOSE MACROS ***************/

[INT TOP]  [TOP:=20]
[MACRO INPUT$WAITING '0FFH']

/******************* DECLARATIONS *******************/

DECLARE (MTS$EXTERNAL, MTS$INTERNAL, EXIT,
    MTS,TERM$BLOCK) LABEL;
DECLARE SAVE DATA (0,0,0);

/******************* ENTRY POINT *******************/

INTERNAL$MTS:  /* INTERNAL ENTRY POINT INTO SERVICE */
               /* MODULE, I.E. ENTRY POINT FROM     */
               /* OTHER MTS ROUTINES                */
    SAVE(2)=(A=[LOCK]);  /* SAVE LOCK VALUE */
    [PARM0]=(HL=DE);  /* SAVE PARM LIST ADDRESS */
    SAVE=(HL=0+SP);    /* SAVE USER SP */
    SP=(HL=.[SVC$STACK$TOP]);
    IF (A=<<C) !CY GOTO MTS;
    A=(A=!C)+1;  /* CONVERT FID TO POSITIVE NR */
    IF (A::3) !CY THEN
        /* INVALID COMMAND - ERROR 1 */
        DO; [ERROR]=(A=1); GOTO EXIT; END;
    H=0; L=A;
    DO CASE HL;
        /*  0 */  DO; NOP; END;
        /* -1 */  DO; CALL [READ$TERMINAL];
                      [ERROR]=A;
                  END;
        /* -2 */  CALL [BUMP];
    END; /* CASE */
    GOTO EXIT;

100H:  /* ADJUST EXTERNAL ENTRY POINT LOCATION */
EXTERNAL$MTS:  /* EXTERNAL ENTRY POINT INTO SERVICE */
               /* MODULE, I.E. ENTRY POINT FROM     */
               /* USER PROGRAMS                     */
    DISABLE;
    SAVE(2)=(A=[LOCK]  & 0FEH);  /* SAVE LOCK VALUE */
    [LOCK]=(A=A \ 01);  /* LOCK OUT SWAPPING */
    ENABLE;
    [PARM0]=(HL=DE);  /* SAVE PARM LIST ADDRESS */
    SAVE=(HL=0+SP);    /* SAVE USER SP */
    SP=(HL=.[SVC$STACK$TOP]);

MTS:  /* SYSTEM AND SERVICE ROUTINES */
    IF (A=C; A::18) !CY THEN
        /* INVALID COMMAND - ERROR 1 */
        DO; [ERROR]=(A=1); GOTO EXIT; END;
    [ERROR]=(A=0);  /* INITIALIZE RETURNED ERROR CODE */
    H=0; L=C;
    DO CASE HL;
        /******** SYSTEM CALLS ***************/
        /*  0 */  CALL [ATTACH];
        /*  1 */  CALL [MTS$MSG];
        /*  2 */  CALL [LOGIN];
        /*  3 */  CALL [PROTECT];
        /*  4 */  CALL [QUIT];
        /*  5 */  CALL [RESTRICT];
        /*  6 */  CALL [SIZE];
        /*  7 */  CALL [UNPROTECT];
        /******** SERVICE CALLS ***************/
        /*  8 */  DO; CALL [TERMINAL$STATUS];
                      [ERROR]=A;
                  END;
        /*  9 */  DO;  CALL [TERMINAL$STATUS];
                     IF (A::[INPUT$WAITING]) ZERO THEN
                         DO; CALL [READ$TERMINAL];
                             [ERROR]=A;
                         END
                     ELSE GOTO TERM$BLOCK;
```

```
            END;
     /* 10 */  CALL [WRITE$TERMINAL];
     /* 11 */  CALL [WRITE$PRINTER];
     /* 12 */  CALL [SELECT$DRIVE];
     /* 13 */  CALL [SET$DMA];
     /* 14 */  CALL [SET$TRACK];
     /* 15 */  CALL [SET$SECTOR];
     /* 16 */  CALL [READ$FLOPPY];
     /* 17 */  CALL [WRITE$FLOPPY];
   END; /* CASE */

EXIT:  /* COMMON EXIT POINT FOR INTERNAL AND EXTERNAL */
   SP=(HL=SAVE);  /* RESTORE USER SP */
   DISABLE;
   [LOCK]=(A=SAVE(2));   /* RESTORE LOCK VALUE */
   A=[ERROR];
   ENABLE;
   RETURN;
   /* END MTS */

TERM$BLOCK:
/*********************************************************/
/* THIS ROUTINE IS CALLED WHEN THE TASK CURRENTLY       */
/* ALLOCATED THE CPU IS BLOCKED FOR TERMINAL I/0. THE */
/* ROUTINE STORES THE CURRENT MACHINE ENVIRONMENT IN   */
/* THE SWAP STACK AND TRANSFERS CONTROL TO THE          */
/* MONITOR FOR SELECTION OF THE NEXT READY TASK.        */
/* CALLED BY: MTS                                        */
/*********************************************************/
   /* SET BIT 5 IN TCT$STATUS */
   DE=.[TCT$STATUS]; A=[TASK]; CALL [INDEX];
   M(HL)=(A=M(HL) \ 20H);
   /* SAVE ENVIRONMENT */
   [SWAP$STACK9]=(HL=SAVE);   /* ONLY USER SP NEEDED */
   GOTO [MONITOR];
   /* END TERM$BLOCK */

EOF
/*********************************************************/
/******************** SYSTEM CALLS ********************/
/*********************************************************/


/************* INTERMODULE LINKAGE MACROS ************/

DECLARE GLOB1 COMMON;
[INT TB MB M2B]  [M2B:=0600H] [TB:=1000H] [MB:=0300H]
[INT S3B]  [S3B:=2700H]
[MACRO MONITOR '[HEX MB + 0AAH]']
[MACRO INDEX '[HEX M2B + 3H]']
[MACRO INDEX2 '[HEX M2B + 0DH]']
[MACRO INDEX4 '[HEX M2B + 18H]']
[MACRO INDEX8 '[HEX M2B + 24H]']
[MACRO GET '[HEX M2B + 38H]']
[MACRO MOVBUF '[HEX M2B + 41H]']
[MACRO MINI$DISK '[HEX M2B + 54H]']
[MACRO SIZE$MSG '[HEX TB + 864H]']
[MACRO STATUS$MSG '[HEX TB + 88CH]']
[MACRO CLEAR$STATUS$LINE '[HEX TB + 827H]']
[MACRO MTS$MSG '[HEX TB + 837H]']
[MACRO PARM0 'GLOB1(10H)']
[MACRO DISK 'GLOB1(12H)']
[MACRO DRIVE 'GLOB1(13H)']
[MACRO ERROR 'GLOB1(14H)']
[MACRO MDBUF 'GLOB1(3FH)']
[MACRO TASK 'GLOB1(241H)']
[MACRO TCT$STATUS 'GLOB1(246H)']
[MACRO TCT$DM 'GLOB1(24AH)']
[MACRO TCT$SIZE 'GLOB1(26AH)']
[MACRO TCT$BOE 'GLOB1(26EH)']
[MACRO TCT$EOE 'GLOB1(276H)']
[MACRO DMT$FLAG 'GLOB1(27EH)']
[MACRO DMT$KEY 'GLOB1(31EH)']
```

```
[MACRO VDC$DRIVE 'GLOB1(3AEH)']
[MACRO CNFG$FILE 'GLOB1(244H)']
[MACRO PRT$CNTRL 'GLOB1(39EH)']
[MACRO READ$BUF '[HEX S3B + 07H]']
[MACRO WRITE$BUF '[HEX S3B + 23H]']

/*************** GENERAL PURPOSE MACROS ***************/

[MACRO WRITE '2']
[MACRO HARDWARE$ERROR '8']

/******************** DECLARATIONS ********************/

DECLARE PLIST DATA (0,0,0FFH,0FFH,0FFH,0FFH);
DECLARE DSK DATA (0);
DECLARE PA DATA (0);

/**************** UTILITY PROCEDURES ****************/

VAL$DRIVE:  PROCEDURE;
/*******************************************************/
/* THIS PROCEDURE VALIDATES THE DRIVE NUMBER INPUT TO */
/* ANY SYSTEM CALL WHICH REQUIRES THAT PARAMETER.     */
/* INPUT: A - DRIVE NUMBER TO BE VALIDATED            */
/* OUTPUT: DRIVE - NUMBER OF FREE DRIVE FOUND         */
/*         ERROR - ERROR CODE                         */
/* CALLED BY: ATTACH                                  */
/*******************************************************/
    IF (A::0FFH) ZERO THEN  /* NO DRIVE SPECIFIED */
        DO;  /* SCAN DRIVE MAP FOR FREE DRIVE */
            DECLARE L1 LABEL;
            DE=.[TCT$DM]; A=[TASK]; CALL [INDEX8];
            B=8;
            REPEAT;
                IF (A=<M(HL)) !CY GOTO L1;
                HL=HL+1;
            UNTIL (B=B-1) ZERO;
            /* NO DRIVE AVAILABLE - ERROR 10 */
            [ERROR]=(A=10); RETURN;
        L1: [DRIVE]=(A=8-B);  /* FREE DRIVE FOUND */
        END
    ELSE IF (A::8) !CY THEN
            /* DRIVE NR > 7 - ERROR 6 */
            DO; [ERROR]=(A=6); RETURN; END;
    END VAL$DRIVE;

VAL$DISK:  PROCEDURE;
/*******************************************************/
/* THIS PROCEDURE VALIDATES THE DISK NUMBER INPUT TO  */
/* ANY SYSTEM CALL WHICH REQUIRES THAT PARAMETER.     */
/* INPUT: A - DISK NUMBER TO BE VALIDATED             */
/* OUTPUT: DISK - NUMBER OF FREE DISK FOUND           */
/*         ERROR - ERROR CODE                         */
/* CALLED BY: ATTACH                                  */
/*******************************************************/
    IF(A::0FFH) ZERO THEN  /* NO DISK SPECIFIED */
        DO;  /* SCAN DISK MAP FOR FREE DISK */
            DECLARE L2 LABEL;
            HL=.[DMT$FLAG]; B=32;
            REPEAT;
                IF (A=>M(HL)) CY  /* DISK AVAILABLE */
                  & (A=>A) !CY    /* NOT IN USE     */
                  & (A=>A) !CY    /* NOT PROTECTED  */
                  & (A=>A) !CY    /* NOT RESTRICTED */
                  THEN GOTO L2;
                HL=HL+1;
            UNTIL (B=B-1) ZERO;
            /* NO DISK AVAILABLE - ERROR 2 */
            [ERROR]=(A=2); RETURN;
        L2: [DISK]=(A=32-B);  /* FREE DISK FOUND */
        END
    ELSE IF (A::32) !CY THEN
            /* DISK NR > 31 - ERROR 4 */
```

```
            DO; [ERROR]=(A=4); RETURN; END
        ELSE
            DO; /* SEE IF SPECIFIED DISK AVAILABLE */
            DE=.[DMT$FLAG]; A=[DISK]; CALL [INDEX];
            IF (A=>M(HL)) !CY THEN
                /* DISK NOT AVAILABLE - ERROR 2 */
                DO; [ERROR]=(A=2); RETURN; END;
            IF (A=>A) CY THEN
                /* DISK IN USE - ERROR 3 */
                DO; [ERROR]=(A=3); RETURN; END;
            END;
    END VAL$DISK;

VAL$KEY:  PROCEDURE;
/*********************************************************/
/* THIS PROCEDURE COMPARES THE KEY INPUT AS A SYSTEM     */
/* CALL PARAMETER WITH THAT ASSOCIATED WITH A SPECI-     */
/* FIED VIRTUAL DISK FILE.                                */
/* INPUT: PARM - VARIABLE HOLDING ADDRESS OF PARM KEY    */
/*        DISK - VIRTUAL DISK FILE NUMBER                 */
/* OUTPUT: ERROR - ERROR CODE                             */
/* CALLED BY: ATTACH                                      */
/*********************************************************/
    DE=.[DMT$KEY]; A=[DISK]; CALL [INDEX4];
    DE=HL; HL=[PARM0];
    DO B=0 BY B=B+1 WHILE (A=4; A::B) !ZERO;
        IF (A=M(DE); A::M(HL)) ZERO
         \ (IF (A::20H) ZERO  & (A=M(HL)-0FFH) ZERO
            THEN CY=1 ELSE CY=0) CY
        THEN  /* CHAR MATCH */
            DO;
            DE=DE+1; HL=HL+1;
            END
        ELSE  /* KEY ERROR - ERROR 5 */
            DO; [ERROR]=(A=5); RETURN; END;
    END; /* WHILE */
    /* KEYS MATCH */
    END VAL$KEY;

CLEAR$FLAG:  PROCEDURE;
/*********************************************************/
/* THIS PROCEDURE RESETS THE IN USE BIT (BIT 1) IN       */
/* THE DMT$FLAG FOR A SPECIFIED VIRTUAL DISK.             */
/* INPUT: B - DISK NUMBER                                 */
/* CALLED BY: ATTACH, LOGIN, CLEAR$DM                     */
/*********************************************************/
    DE=.[DMT$FLAG]; A=B  & 1FH; CALL [INDEX];
    M(HL)=(A=M(HL)  & 0FDH);
    END CLEAR$FLAG;

CLEAR$DM:  PROCEDURE;
/*********************************************************/
/* THIS PROCEDURE RESETS ALL ENTRIES IN THE TCT$DM       */
/* ASSOCIATED WITH THE CURRENT VALUE OF TASK.             */
/* CALLED BY: LOGIN, QUIT                                 */
/*********************************************************/
    DE=.[TCT$DM]; A=[TASK]; CALL [INDEX8];
    C=8;
    REPEAT;
        B=M(HL); M(HL)=0;
        IF (A=<B) CY  & (A=<A) !CY THEN
            DO;
            STACK=HL;
            STACK=BC;
            CALL CLEAR$FLAG;
            BC=STACK;
            HL=STACK;
            END;
        HL=HL+1;
    UNTIL (C=C-1) ZERO;
    END CLEAR$DM;

/****************** SYSTEM ROUTINES ******************/
```

```
BUMP:  PROCEDURE;
/**********************************************************/
/* THIS INTERNAL SYSTEM CALL DELETES THE CURRENT TASK */
/* FROM THE SYSTEM.                                     */
/* ARGUMENTS:                                           */
/*   (1) FID = -2                                       */
/*   (2) PARM = NONE                                    */
/* VALUE: NONE                                          */
/* CALLED BY: INTERNAL$MTS, QUIT                        */
/**********************************************************/
    /* WRITE MINI-DISK BUFFER IF NECESSARY */
    CALL [WRITE$BUF];
    IF (A=[ERROR] \ A) !ZERO THEN
        DO; E=[HARDWARE$ERROR]; CALL [MTS$MSG]; END;
    /* CLEAR STATUS LINE */
    A=[TASK]; CALL [CLEAR$STATUS$LINE];
    /* CLEAR TCT */
    CALL CLEAR$DM;
    DE=.[TCT$SIZE]; A=[TASK]; CALL [INDEX];
    M(HL)=32;  /* SIZE = 16K */
    HL=BC+(DE=.[TCT$STATUS]);
    M(HL)=0;  /* RESET STATUS BYTE */
    IF (A=<[PRT$CNTRL]) CY THEN /* PRINTER IN USE */
      DO;
        A=>A; B=(A=A & 03H);
        IF (A=[TASK]; A::B) ZERO THEN
        DO;  /* THIS TASK HAS PRINTER CONTROL */
            IF (A=<[PRT$CNTRL]; A=<A) CY THEN
            /* RESET PRINTER CONTROL BYTE */
            [PRT$CNTRL]=(A=0);
        END;
      END;
    END BUMP;

ATTACH:  PROCEDURE;
/**********************************************************/
/* SIMULATE THE PHYSICAL OPERATION OF LOADING DISK      */
/* <DISK NR> INTO DRIVE <DRIVE LTR>.                    */
/* ARGUMENTS:                                           */
/*  (1) FID = 0                                         */
/*  (2) PARM = BASE ADDRESS OF PARAMETER LIST           */
/*       BYTE 0: DRIVE NUMBER (0-7)                     */
/*       BYTE 1: DISK NUMBER (0-31)                     */
/*       BYTES 2-5: PROTECTION KEY (0-4 CHARS)          */
/* VALUE: ERROR CODE                                    */
/* CALLED BY: LOGIN, MTS                                */
/**********************************************************/
    BC=(HL=[PARM0]); [DRIVE]=(A=M(BC));
    /* VALIDATE DRIVE NR */
    CALL VAL$DRIVE; IF (A=[ERROR] \ A) !ZERO RETURN;
    HL=[PARM0]+1; [DISK]=(A=M(HL));
    /* VALIDATE DISK NR */
    CALL VAL$DISK; IF (A=[ERROR] \ A) !ZERO RETURN;
    DE=.[DMT$FLAG]; A=[DISK]; CALL [INDEX];
    IF (A=M(HL)  & 04H) !ZERO THEN  /* DISK PROTECTED */
    DO;
        DE=.[DMT$FLAG];  A=[DISK];  CALL [INDEX];
        IF (A=M(HL) & 08H) !ZERO THEN
        DO;   /* READ ONLY DISK */
        /*  SET UP READ ONLY BIT  */
            M(HL)=(A=M(HL) \ 02H);
            [DISK]=(A=[DISK] \ 40H); /* SET TCT$DM BIT 6 */
        END
        ELSE
        DO;  /* VALIDATE KEY */
            [PARM0]=(HL=[PARM0]+1,+1);
            CALL VAL$KEY;
            IF (A=[ERROR] \ A) !ZERO RETURN;
        END;
    END;
    /* MODIFY DMT$FLAG */
    DE=.[DMT$FLAG]; A=[DISK]  & 1FH; CALL [INDEX];
```

```
    M(HL)=(A=M(HL) \\ 02H);  /* SET DMT$FLAG BIT 1 */
    /* MODIFY TCT$DM */
    DE=.[TCT$DM]; A=[TASK]; CALL [INDEX8];
    DE=HL; A=[DRIVE]; CALL [INDEX];
    B=M(HL);
    M(HL)=(A=[DISK] \ 80H);  /* SET TCT$DM BIT 7 */
    /* RESET OLD DISK'S IN USE BIT */
    IF (A=B  & 40H) ZERO CALL CLEAR$FLAG;
    /* DISPLAY STATUS MSG */
    B=(A=[DRIVE]); C=(A=[DISK]  & 1FH);
    IF (A=[DISK]  &40H) !ZERO THEN A=72H ELSE A=' ';
    CALL [STATUS$MSG];
    END ATTACH;

LOGIN:  PROCEDURE;
/*********************************************************/
/* THIS SYSTEM CALL NOTIFIES MTS THAT THE REQUESTING  */
/* TERMINAL IS NOW ACTIVE, AND SIMULATES THE PHYSICAL */
/* COLD-START BOOTSTRAP OPERATION. THE BOOTSTRAP LOAD */
/* TAKES PLACE FROM VIRTUAL DRIVE A. THE VIRTUAL DISK */
/* ATTACHED TO THIS DRIVE MAY BE SPECIFIED IN THE     */
/* PARAMETER LIST, OR DISK0 WILL BE ASSUMMED AS THE   */
/* DEFAULT.                                           */
/* ARGUMENTS:                                         */
/*  (1) FID = 2                                       */
/*  (2) PARM = BASE ADDRESS OF PARAMETER LIST         */
/*        BYTE 0: DISK NUMBER (0-31)                  */
/*        BYTES 1-4: PROTECTION KEY (0-4 CHARS)       */
/* VALUE: ERROR CODE                                  */
/*********************************************************/
    /* WRITE MINI-DISK BUFFER IF NECESSARY */
    CALL [WRITE$BUF];
    IF (A=[ERROR] \ A) !ZERO RETURN;
    /* RE-INITIALIZE TCT */
    CALL CLEAR$DM;
    /* PROCESS DISK PARAMETER */
    IF (HL=[PARM0]; A=M(HL); A::0FFH) !ZERO THEN
    DO; /* DISK SPECIFIED */
        BC=5; DE=HL; HL=.PLIST(1);
        CALL [MOVBUF]; [PARM0]=(HL=.PLIST);
    END
    ELSE
    DO; /* ASSUME DISK 0 */
        PLIST(1)=(A=0); PLIST(2)=(A='M');
        PLIST(3)=(A='T'); PLIST(4)=(A='S');
        PLIST(5)=(A=' ');
        [PARM0]=(HL=.PLIST);
    END;
    CALL ATTACH;
    IF (A=[ERROR] \ A) !ZERO RETURN;
    B=0; C=(A=[TASK]);
    HL=BC+(DE=.[TCT$STATUS]);
    M(HL)=1;  /* SET BIT 0 */
    /* DISPLAY SIZE MSG */
    DE=.[TCT$SIZE]; A=[TASK]; CALL [INDEX];
    CY=0; A=>M(HL); CALL [SIZE$MSG];
    END LOGIN;

UPDATE$CNFG$FILE:  PROCEDURE;
/*********************************************************/
/* THIS ROUTINE PROVIDES INDEXING INTO THE MTS        */
/* CONFIGURATION FILE AND PERMITS MODIFICATION OF     */
/* OF THE CONFIGURATION FILE WHICH IS NECESARRY       */
/* WHEN PROTECTING, UNPROTECTING OR RESTRICTING       */
/* USER FILES.                                        */
/* CALLED BY:  PROTECT, UNPROTECT, RESTRICT           */
/*********************************************************/
    PA=A;  CALL [WRITE$BUF];
    BC=(HL=[CNFG$FILE]);
    CALL [READ$BUF];
    DE=.[MDBUF]; A=DSK;  CALL [INDEX8];  DE=HL;
    A=DSK;  CALL [INDEX4];  B=0;  C=(A=DSK);
    HL=HL+BC;  HL=HL+(DE=8);
```

```
    STACK=HL; BC=(HL=[PARM0]); HL=STACK; BC=BC+1; D=0;
    REPEAT;
        A=M(BC); IF (A::0FFH) ZERO THEN A=20H;
        M(HL)=A;  /* CHANGE FF PARAMETER TO SPACE */
        BC=BC+1; HL=HL+1; D=(A=D+1);
    UNTIL (A::4) ZERO;
    M(HL)=(A=PA);  /* STORE PROTECTION ATTRIBUTE */
    [VDC$DRIVE]=(A=[VDC$DRIVE] \ 80H);
    END UPDATE$CNFG$FILE;

PROTECT:  PROCEDURE;
/**********************************************************/
/* THIS SYSTEM CALL ADDS THE READ/WRITE PROTECTION       */
/* ATTRIBUTE TO A SPECIFIED VIRTUAL DISK FILE.           */
/* ARGUMENTS:                                            */
/*  (1) FID = 3                                          */
/*  (2) PARM = BASE ADDRESS OF PARAMETER LIST            */
/*       BYTE 0: DISK NUMBER (0-31)                      */
/*       BYTES 1-4: PROTECTION KEY (0-4 CHARS)           */
/* VALUE: ERROR CODE                                     */
/**********************************************************/
    IF (HL=[PARM0]; DSK=(A=M(HL)); A::32) !CY THEN
    DO;  /* DISK # > 31 */
        [ERROR]=(A=4);
        RETURN;
    END
    ELSE
    IF (HL=HL+1; A=M(HL); A::0FFH) ZERO THEN
    DO;  /* KEY ERROR */
        [ERROR]=(A=5);
        RETURN;
    END
    ELSE
    IF (DE=.[DMT$FLAG]; A=DSK; CALL [INDEX];
        A=M(HL); A=>A; A=>A; A=>A) CY THEN
    DO;  /* DISK ALREADY PROTECTED */
        [ERROR]=(A=5);  RETURN;
    END
    ELSE
    DO;
        DE=.[DMT$KEY]; A=DSK; CALL [INDEX4];
        BC=HL; HL=[PARM0]+1; D=0;
        REPEAT;  /* UPDATE DMT$KEY */
            A=M(HL); IF (A::0FFH) ZERO THEN A=20H;
            M(BC)=A; BC=BC+1; HL=HL+1; D=(A=D+1);
        UNTIL (A::4) ZERO;
        DE=.[DMT$FLAG]; A=DSK; CALL [INDEX];
        M(HL)=(A=M(HL) \ 04H);  /* SET PROTECT BIT */
        A='P'; CALL UPDATE$CNFG$FILE;
    END;
    END PROTECT;

QUIT:  PROCEDURE;
/**********************************************************/
/* THIS SYSTEM CALL NOTIFIES MTS THAT THE REQUESTING    */
/* TERMINAL IS NO LONGER ACTIVE.                         */
/* ARGUMENTS:                                            */
/*  (1) FID = 4                                          */
/*  (2) PARM = NONE                                      */
/* VALUE: NONE                                           */
/**********************************************************/
    CALL BUMP;
    GOTO [MONITOR];
    END QUIT;

RESTRICT:  PROCEDURE;
/**********************************************************/
/* THIS SYSTEM CALL ADDS THE READ RESTRICTION ATTRI-    */
/* BUTE TO A SPECIFIED PROTECTED VIRTUAL DISK FILE.      */
/* ARGUMENTS:                                            */
/*  (1) FID = 5                                          */
/*  (2) PARM = BASE ADDRESS OF PARAMETER LIST            */
/*       BYTE 0: DISK NUMBER (0-31)                      */
```

```
/*        BYTES 1-4: PROTECTION KEY (0-4 CHARS)         */
/* VALUE: ERROR CODE                                     */
/*******************************************************/
    IF (HL=[PARM0];  DSK=(A=M(HL));   A::32) !CY THEN
    DO;  /* DISK > 31 */
        [ERROR]=(A=4); RETURN;
    END;
    IF (DE=.[DMT$FLAG];  A=DSK;  CALL [INDEX];
        A=M(HL);  A=>A;  A=>A) CY THEN
    DO;  /*  DISK IN USE */
        [ERROR]=(A=3);  RETURN;
    END;
    IF (DE=.[DMT$FLAG];  A=DSK;  CALL [INDEX];
        A=M(HL);  A=>A;  A=>A;  A=>A) !CY THEN
    DO;  /* DISK NOT PROTECTED */
        [ERROR]=(A=5);  RETURN;
    END;
    DE=.[DMT$KEY]; A=DSK; CALL [INDEX4];
    STACK=HL; BC=(HL=[PARM0]); BC=BC+1; HL=STACK; E=0;
    REPEAT;
        IF (A=M(BC);  IF (A::0FFH) ZERO THEN A=20H;
            A::M(HL)) !ZERO THEN
        DO;  /*  KEY DOES NOT MATCH */
            [ERROR]=(A=5);  RETURN;
        END;
        HL=HL+1;  BC=BC+1;  E=(A=E+1);
    UNTIL (A::4) ZERO;
    /*  SET RESTRICTION BIT IN DMT$FLAG  */
    DE=.[DMT$FLAG];  A=DSK;  CALL [INDEX];
    M(HL)=(A=M(HL) \ 08H);
    A='R';  CALL UPDATE$CNFG$FILE;
    END RESTRICT;

SIZE:  PROCEDURE;
/*******************************************************/
/* THIS SYSTEM CALL SETS THE SIZE OF THE USER'S SWAP */
/* IMAGE.                                              */
/* ARGUMENTS:                                          */
/*   (1) FID = 6                                       */
/*   (2) PARM = REQUESTED SIZE IN KILOBYTES            */
/* VALUE: ERROR CODE                                   */
/*******************************************************/
    /* COMPARE REQUESTED SIZE WITH MAX SIZE */
    IF (A=[PARM0]; A::49) !CY THEN
        /* OUT OF BOUNDS - INVALID CMD ERROR 1 */
        DO; [ERROR]=(A=1); RETURN; END;
    /* COMPARE REQUESTED SIZE WITH SWAP FILE SIZE */
    DE=.[TCT$BOE]; A=[TASK]; CALL [INDEX2];
    CALL [GET]; B=0; C=(A=<<[PARM0]);
    HL=BC+DE; STACK=HL;  /* SAVE SUM */
    DE=.[TCT$EOE]; A=[TASK]; CALL [INDEX2];
    CALL [GET]; BC=BC+1;
    DE=STACK;  /* RESTORE SUM */
    A=C-E; A=B--D;
    IF MINUS THEN
        /* OUT OF BOUNDS - INVALID CMD ERROR 1 */
        DO; [ERROR]=(A=1); RETURN; END;
    /* DISPLAY SIZE MSG */
    A=[PARM0]; CALL [SIZE$MSG];
    /* UPDATE TCT */
    DE=.[TCT$SIZE]; A=[TASK]; CALL [INDEX];
    M(HL)=(A=<<[PARM0]);
    END SIZE;

UNPROTECT:  PROCEDURE;
/*******************************************************/
/* THIS SYSTEM CALL DELETES THE READ RESTRICTION AND */
/* READ/WRITE PROTECTION ATTRIBUTES FROM A SPECIFIED */
/* VIRTUAL DISK FILE.                                  */
/* ARGUMENTS:                                          */
/*   (1) FID = 7                                       */
/*   (2) PARM = BASE ADDRESS OF PARAMETER LIST         */
/*        BYTE 0: DISK NUMBER (0-31)                   */
```

```
/*        BYTES 1-4: PROTECTION KEY (0-4 CHARS)          */
/* VALUE: ERROR CODE                                      */
/*********************************************************/
    IF (HL=[PARM0];  DSK=(A=M(HL));  A::32) !CY THEN
    DO;  /* DISK # > 31 */
        [ERROR]=(A=4);
        RETURN;
    END
    ELSE
    IF (HL=HL+1; A=M(HL); A::0FFH) ZERO THEN
    DO;  /* KEY ERROR */
        [ERROR]=(A=5);
        RETURN;
    END
    ELSE
    DO;  /* CHECK IF DMT$KEY MATCHES KEY */
        DE=.[DMT$KEY]; A=DSK; CALL [INDEX4];
        STACK=HL; BC=(HL=[PARM0]); BC=BC+1; HL=STACK; E=0;
        REPEAT;
            IF (A=M(BC); IF (A::0FFH) ZERO THEN A=20H;
                A::M(HL)) !ZERO THEN
            DO;  /* DMT$KEY NOT = KEY */
                [ERROR]=(A=5); RETURN;
            END;
            HL=HL+1;   BC=BC+1;   E=(A=E+1);
        UNTIL (A::4) ZERO;
    END;
    /* SET DMT$KEY = 20H,20H,20H,20H */
    DE=.[DMT$KEY];  A=DSK;  CALL [INDEX4];  E=0;
    REPEAT;
        M(HL)=(A=20H);  HL=HL+1;  E=(A=E+1);
    UNTIL (A::4) ZERO;
    DE=.[DMT$FLAG];  A=DSK;  CALL [INDEX];
    /* RESET BIT 3 AND 4 OF DMT$FLAG */
    M(HL)=(A=M(HL) & 03H);
    A= ' ';  CALL UPDATE$CNFG$FILE;
    END UNPROTECT;


EOF
/*********************************************************/
/******************** SERVICE CALLS ********************/
/*********************************************************/


/************** INTERMODULE LINKAGE MACROS ************/

DECLARE GLOB1 COMMON;
[INT M2B]   [M2B:=0600H]
[MACRO INDEX '[HEX M2B + 3H]']
[MACRO INDEX2 '[HEX M2B + 0DH]']
[MACRO INDEX8 '[HEX M2B + 24H]']
[MACRO GET '[HEX M2B + 38H]']
[MACRO MOVBUF '[HEX M2B + 41H]']
[MACRO MINI$DISK '[HEX M2B + 54H]']
[MACRO READ$PRT$BUF '[HEX M2B + 9EH]']
[MACRO PRT$STAT 'GLOB1(0)']
[MACRO BUF$PTR 'GLOB1(1)']
[MACRO PARM0 'GLOB1(10H)']
[MACRO PARM1 'GLOB1(11H)']
[MACRO DISK 'GLOB1(12H)']
[MACRO DRIVE 'GLOB1(13H)']
[MACRO ERROR 'GLOB1(14H)']
[MACRO MDBUF 'GLOB1(3FH)']
[MACRO MDSAD0 'GLOB1(23FH)']
[MACRO MDSAD1 'GLOB1(240H)']
[MACRO TASK 'GLOB1(241H)']
[MACRO TCT$DM 'GLOB1(24AH)']
[MACRO DMT$BOE 'GLOB1(29EH)']
[MACRO DMT$EOE 'GLOB1(2DEH)']
[MACRO PRT$BOE 'GLOB1(39FH)']
[MACRO PRT$EOE 'GLOB1(3A1H)']
[MACRO PRT$SEC 'GLOB1(3A3H)']
```

```
[MACRO VDC$DRIVE 'GLOB1(3AEH)']
[MACRO VDC$BOE 'GLOB1(3AFH)']
[MACRO VDC$EOE0 'GLOB1(3B1H)']
[MACRO VDC$EOE1 'GLOB1(3B2H)']
[MACRO VDC$SECTOR 'GLOB1(3B3H)']
[MACRO VDC$TRACK 'GLOB1(3B4H)']
[MACRO VDC$DMA 'GLOB1(3B5H)']
[MACRO PRT$CNTRL 'GLOB1(39EH)']

/*************** GENERAL PURPOSE MACROS ***************/

[MACRO READ '1']
[MACRO WRITE '2']
[MACRO CNTRL$Z  '1AH']
[MACRO PRT$RDY '0B0H']
[MACRO CNTRL$R '12H']

/***************** MODULE DECLARATION ******************/

DECLARE CHAR DATA (0);

/***************** UTILITY PROCEDURES *****************/

READ$BUF:  PROCEDURE;
/*********************************************************/
/* THIS PROCEDURE READS A SPECIFIED MINI-DISK SECTOR    */
/* INTO MDBUF AND UPDATES MDSAD.                         */
/* INPUT: BC - MINI-DISK SECTOR NUMBER                   */
/* OUTPUT: ERROR - ERROR CODE                            */
/* CALLED BY: MAP, UPDATE$CNFG$FILE                      */
/*********************************************************/
    DE=.[MDBUF]; L=[READ]; CALL [MINI$DISK];
    IF (A::0) !ZERO THEN
        /* HARDWARE ERROR - ERROR 8 */
        DO; [ERROR]=(A=8); RETURN; END;
    [MDSAD0]=(HL=BC);
    END READ$BUF;

WRITE$BUF:  PROCEDURE;
/*********************************************************/
/* THIS PROCEDURE CHECKS THE MODIFICATION BIT IN        */
/* VDC$DRIVE TO DETERMINE IF THE CONTENTS OF MDBUF      */
/* HAVE BEEN ALTERED. IF SO, THE BUFFER IS WRITTEN TO   */
/* THE MINI-DISK AND THE MOD BIT IS RESET.               */
/* CALLED BY: BUMP, MAP, LOGIN, UPDATE$CNFG$FILE,        */
/*            SELECT$DRIVE                               */
/*********************************************************/
    IF (A=<[VDC$DRIVE]) !CY RETURN;
    BC=(HL=[MDSAD0]); DE=.[MDBUF];
    L=[WRITE]; CALL [MINI$DISK];
    IF (A::0) !ZERO THEN
        /* HARDWARE ERROR - ERROR 8 */
        DO; [ERROR]=(A=8); RETURN; END;
    [VDC$DRIVE]=(A=[VDC$DRIVE]  8 7FH);
    END WRITE$BUF;

MAP:  PROCEDURE;
/*********************************************************/
/* THIS PROCEDURE CALCULATES THE RELATIVE OFFSET OF A   */
/* SPECIFIED VIRTUAL FLOPPY DISK SECTOR IN THE MINI-    */
/* DISK FILE AND THE BASE ADDRESS OF THAT SECTOR IN     */
/* THE MINI-DISK BUFFER AFTER THE FILE IS READ. THEN    */
/* IT CALCULATES THE ACTUAL MINI-DISK SECTOR NUMBER     */
/* CONTAINING THE ADDRESSED FLOPPY DISK SECTOR, AND     */
/* COMPARES IT WITH THE CURRENT CONTENTS OF MDBUF. IF   */
/* THE TWO ARE NOT EQUAL, THE OLD BUFFER IS WRITTEN     */
/* TO THE MINI-DISK AND THE NEWLY CALCULATED SECTOR     */
/* NUMBER READ IN TO REFILL THE BUFFER. A COMPARISON    */
/* IS ALSO MADE BETWEEN THE CALCULATED MINI-DISK        */
/* SECTOR NUMBER AND THE FILE'S EOE VALUE TO ENSURE     */
/* THAT THE SPECIFIED VIRTUAL DISK SECTOR ADDRESS IS    */
/* WITHIN THE BOUNDS OF THE FILE.                       */
/* INPUT: VDC$TRACK - FLOPPY DISK TRACK NUMBER          */
```

```
/*        VDC$SECTOR - FLOPPY DISK SECTOR NUMBER       */
/* OUTPUT: BC = 128 = FLOPPY DISK SECTOR SIZE          */
/*         HL = BASE ADDRESS OF FLOPPY DISK SECTOR     */
/*              IN BUFFER                              */
/* CALLED BY: READ$FLOPPY, WRITE$FLOPPY                */
/*******************************************************/
    DECLARE BUFAD DATA (0,0);
    DECLARE SECNR DATA (0,0);
    D=(A=[VDC$TRACK]);
    E=(A=[VDC$SECTOR]);
    /* MULTIPLY TRACK # BY 26; ADD SECTOR #; SUBTRACT 1 */
    B=0; CY=0;
    A=<D; L=A; /* 2 * TRACK;  SAVE IN L REG */
    CY=0; A=<A; /* 4 * TRACK */
    IF CY THEN B=1;  /* SAVE HIGH ORDER BITS IN B */
    A=A+L; C=A;  A=L; /* SAVE 2 * TRACK IN ACCUM. */
    HL=0; HL=HL+BC; HL=HL+BC; HL=HL+BC; HL=HL+BC;
    /* HL NOW CONTAINS 24 * THE TRACK # */
    B=0; C=A; HL=HL+BC;  /* HL CONTAINS 26 * TRACK # */
    C=E; HL=HL+BC; HL=HL-1; /* SECTOR # ADDED ON AND SUB 1 */
    /* DIVIDE CONTENTS OF HL BY 4 */
    DE=HL; /* SAVE 26 * TRACK + SECTOR # -1 */
    CY=0; D=(A=>D);  E=(A=>E);
    CY=0;  D=(A=>D);  A=>E;
    B=D;  C=A;  /* BC HAS (26 * SECTOR # -1) / 4 */
    /* COMPUTE BUFFER OFFSET */
    IF (A=[VDC$TRACK]; A=A & 01H) ZERO THEN
        A=[VDC$SECTOR] /* TRACK IS EVEN */
    ELSE
        A=[VDC$SECTOR],+1,+1; /* TRACK IS ODD */
TEST:
    IF (A::5) !CY THEN
        DO;
        A=A-4;
        GOTO TEST;
        END;
    L=A-1; H=0;
    DO CASE HL;
        DE=0;
        DE=128;
        DE=256;
        DE=384;
    END;

                /* DE = RELATIVE BUFFER ADDRESS */
    BUFAD=(HL=.[MDBUF], +DE);
    /* CALCULATE AND SAVE NEW MINI-DISK SECTOR NR */
    SECNR=(HL=[VDC$BOE] + BC);
    /* COMPARE NEW SECTOR NR WITH VDC$EOE */
    A=[VDC$EOE0]-L; A=[VDC$EOE1]--H;
    IF MINUS THEN
        /* OUT OF BOUNDS - ERROR 12 */
        DO; [ERROR]=(A=12); RETURN; END;
    /* COMPARE NEW SECTOR NR WITH MDSAD */
    IF (A=[MDSAD0]; A::L)  !ZERO THEN
        CY=0
    ELSE IF (A=[MDSAD1]; A::H) !ZERO THEN
            CY=0
        ELSE CY=1;
    /* WRITE OLD SECTOR AND READ NEW IF NECESSARY */
    IF !CY THEN
        DO;  /* NOT EQUAL */
        CALL WRITE$BUF;
        IF (A=[ERROR] \ A) !ZERO RETURN;
        BC=(HL=SECNR); CALL READ$BUF;
        IF (A=[ERROR] \ A) !ZERO RETURN;
        END;
    /* SET UP REGISTERS FOR RETURN */
    BC=128; DE=(HL=BUFAD);
    END MAP;

WRITE$PRT$BUF:  PROCEDURE;
/*******************************************************/
```

```
/* THIS ROUTINE IS CALLED BY WRITE$PRINTER WHEN THE     */
/* PRINT BUFFER LOCATED BETWEEN 100 - 300H IN MEMORY    */
/* IS FULL OR WHEN A CONTROL Z (EOF INDICATOR) HAS      */
/* BEEN WRITTEN INTO THE BUFFER.  IN BOTH CASES THE     */
/* CONTENTS OF THE BUFFER IS WRITTEN ONTO THE MINI-     */
/* DISK IN MTSPRT.                                      */
/* CALLED BY: WRITE$PRINTER                             */
/*******************************************************/
    HL=[PRT$SEC]; BC=HL;
    DE=100H; L=[WRITE];  /* SET UP FOR CALL TO MINIDISK */
    CALL [MINI$DISK];
    IF(A::0) !ZERO THEN
    DO;
        [ERROR]=(A=8);
        RETURN;
    END;
    HL=[PRT$EOE];
    IF (A=H; A::B) CY & (A=L; A::C) CY THEN
    DO; /* EXCEEDED EOE OF MTS$PRT$FILE */
        [ERROR]=(A=12);
    END
    ELSE
    DO;
        BC=BC+1;
        [PRT$SEC]=(HL=BC);
        /*  PRT$SEC INCREMENTED  */
    END;
    END WRITE$PRT$BUF;

/****************** SERVICE ROUTINES ******************/

WRITE$PRINTER:  PROCEDURE;
/*******************************************************/
/* THIS SERVICE CALL IS MADE TO WRITE CHARACTERS INTO */
/* THE PRINTER BUFFER WHICH IS LOCATED BETWEEN 100 -  */
/* 300H IN MEMORY.                                     */
/* ARGUMENTS:                                          */
/*  (1) FID = 11                                       */
/*  (2) PARM = ASCII CHARACTER                         */
/* VALUE: ERROR CODE                                   */
/*******************************************************/
    CHAR=(A=E); /* SAVE CHARACTER */
    IF (A=[PRT$CNTRL]; A::0) ZERO THEN
    DO;   /* PRINTER NOT IN USE */
        IF ([PRT$STAT]=(A=IN(8AH)); A::[PRT$RDY]) !ZERO THEN
        DO;
            [ERROR]=(A=7);
            RETURN;
        END;
        /* SET PRINTER IN USE BIT */
        [PRT$CNTRL]=(A=[TASK] \ 80H);
        [PRT$SEC]=(HL=[PRT$BOE]);
        [BUF$PTR]=(HL=100H);
    END
    ELSE
    DO;   /* CHECK IF CURRENT TASK OWNS PRINTER */
        B=(A=A & 3); /* B = TASK WHICH HAS PRINTER */
        IF (A=[TASK];  A::B) !ZERO THEN
        DO;  /* TASK DOES NOT OWN PRINTER */
            [ERROR]=(A=11);
            RETURN;
        END
        ELSE
        IF (A=[PRT$CNTRL]; A=<A; A=<A) CY THEN
        DO;  /*  PRINTER IS PRINTING A FILE  */
            [ERROR]=(A=11);
            RETURN;
        END;
    END;
    IF (A=CHAR; A::[CNTRL$Z]) !ZERO & (A::[CNTRL$R]) !ZERO THEN
    DO;  /* FILL BUFFER */
        HL=[BUF$PTR];
        IF (A=3; A::H) ZERO & (A=0; A::L) ZERO THEN
```

```
        DO; /* BUFFER FULL - WRITE TO DISK */
            CALL WRITE$PRT$BUF;
            [BUF$PTR]=(HL=100H);
        END;
        HL=[BUF$PTR];
        M(HL)=(A=CHAR);
        [BUF$PTR]=(HL=[BUF$PTR], +1);
    END
    ELSE
    DO;  /* END OF FILE CONDITION, SET UP FOR PRINTING */
        IF (A::[CNTRL$Z]) ZERO THEN
        DO; /* EOF */
            HL=[BUF$PTR];
            M(HL)=(A=CHAR);
            CALL WRITE$PRT$BUF;
        END;
        [PRT$SEC]=(HL=[PRT$BOE]);
        [BUF$PTR]=(HL=100H);
        CALL [READ$PRT$BUF];
        DISABLE;
        [PRT$CNTRL]=(A=[PRT$CNTRL] \ 40H);
        /* START PRINTING */
        OUT(8AH)=(A=0);
        ENABLE;
    END;
    END WRITE$PRINTER;

SELECT$DRIVE:  PROCEDURE;
/*********************************************************/
/* THIS SERVICE CALL SELECTS THE VIRTUAL FLOPPY DISK    */
/* DRIVE TO BE USED FOR SUBSEQUENT FLOPPY DISK          */
/* ACCESSES.                                            */
/* ARGUMENTS:                                           */
/*  (1) FID = 12                                        */
/*  (2) PARM = DRIVE NUMBER (0-7)                       */
/* VALUE: ERROR CODE                                    */
/*********************************************************/
    [DRIVE]=(A=[PARM0]);
    /* VALIDATE DRIVE NUMBER */
    IF (A::8) !CY THEN
        /* DRIVE NR > 7 - ERROR 6 */
        DO; [ERROR]=(A=6); RETURN; END;
    /* VALIDATE THAT DRIVE IN USE */
    DE=.[TCT$DM]; A=[TASK]; CALL [INDEX8];
    DE=HL; A=[DRIVE]; CALL [INDEX];
    IF (A=<M(HL)) !CY THEN
        /* DRIVE NOT AVAIL - ERROR 10 */
        DO; [ERROR]=(A=10); RETURN; END;
    STACK=HL;
    CALL WRITE$BUF;
    HL=STACK;
    /* UPDATE VDC BLOCK */
    [DISK]=(A=M(HL)  & 1FH);
    IF (A=M(HL)  & 40H) !ZERO THEN  /* READ ONLY */
        [DRIVE]=(A=[DRIVE] \ 40H);  /* SET BIT 6 */
    DE=.[DMT$BOE]; A=[DISK]; CALL [INDEX2];
    CALL [GET]; [VDC$BOE]=(HL=BC);
    DE=.[DMT$EOE]; A=[DISK]; CALL [INDEX2];
    CALL [GET]; [VDC$EOE0]=(HL=BC);
    [VDC$DRIVE]=(A=[DRIVE]);
    END SELECT$DRIVE;

SET$DMA:  PROCEDURE;
/*********************************************************/
/* THIS SERVICE CALL SETS THE ADDRESS OF THE 128 BYTE */
/* DMA BUFFER TO BE USED IN SUBSEQUENT VIRTUAL FLOPPY */
/* DISK ACCESSES.                                     */
/* ARGUMENTS:                                         */
/*  (1) FID = 13                                      */
/*  (2) PARM = DMA ADDRESS                            */
/* VALUE: ERROR CODE                                  */
/*********************************************************/
    A=[PARM0]-0; A=[PARM1]--40H;
```

```
    IF MINUS THEN
        /* OUT OF BOUNDS - ERROR 12 */
        DO; [ERROR]=(A=12); RETURN; END;
    [VDC$DMA]=(HL=[PARM0]);
    END SET$DMA;

SET$TRACK:  PROCEDURE;
/**********************************************************/
/* THIS SERVICE CALL SETS THE TRACK NUMBER TO BE USED */
/* IN SUBSEQUENT VIRTUAL FLOPPY DISK ACCESSES.        */
/* ARGUMENTS:                                         */
/*  (1) FID = 14                                      */
/*  (2) PARM = TRACK NUMBER                           */
/* VALUE: NONE                                        */
/**********************************************************/
    A=[PARM0];
    IF (A::77) !CY THEN
        /* OUT OF BOUNDS - ERROR 12 */
        DO; [ERROR]=(A=12); RETURN; END;
    [VDC$TRACK]=A;
    END SET$TRACK;

SET$SECTOR:  PROCEDURE;
/**********************************************************/
/* THIS SERVICE CALL SETS THE SECTOR NUMBER TO BE     */
/* USED IN SUBSEQUENT VIRTUAL FLOPPY DISK ACCESSES.   */
/* ARGUMENTS:                                         */
/*  (1) FID = 15                                      */
/*  (2) PARM = SECTOR NUMBER (1-26)                   */
/* VALUE: ERROR CODE                                  */
/**********************************************************/
    A=[PARM0];
    IF (A::27) !CY  \ (A::0) ZERO THEN
        /* OUT OF BOUNDS - ERROR 12 */
        DO; [ERROR]=(A=12); RETURN; END;
    [VDC$SECTOR]=A;
    END SET$SECTOR;

READ$FLOPPY:  PROCEDURE;
/**********************************************************/
/* THIS SERICE CALL SIMULATES READING FROM A FLOPPY   */
/* DISK. THE VIRTUAL DISK SECTOR AND TRACK SPECIFIED  */
/* IN THE VDC BLOCK IS READ FROM THE SPECIFIED        */
/* VIRTUAL DRIVE INTO A 128 BYTE BUFFER IN THE USER'S */
/* SWAP AREA.                                         */
/* ARGUMENTS:                                         */
/*  (1) FID =16                                       */
/*  (2) PARM = NONE                                   */
/*      DRIVE, SECTOR, TRACK, AND DMA ADDRESS MUST    */
/*      HAVE BEEN PREVIOUSLY SET BY CALLS TO THE      */
/*      APPROPRIATE PROCEDURES.                       */
/* VALUE: ERROR CODE                                  */
/**********************************************************/
    CALL MAP;
    IF (A=[ERROR] \ A) !ZERO RETURN;
    DE=HL; HL=[VDC$DMA]; CALL [MOVBUF];
    END READ$FLOPPY;

WRITE$FLOPPY:  PROCEDURE;
/**********************************************************/
/* THIS SERVICE CALL SIMULATES WRITING TO A FLOPPY    */
/* DISK. A 128 BYTE BUFFER IN THE USER SWAP AREA IS   */
/* WRITTEN TO THE VIRTUAL TRACK, SECTOR, AND DRIVE    */
/* SPECIFIED IN THE VDC BLOCK.                        */
/* ARGUMENTS:                                         */
/*  (1) FID = 17                                      */
/*  (2) PARM = NONE                                   */
/*      DRIVE, SECTOR, TRACK, AND DMA ADDRESS MUST    */
/*      HAVE BEEN PREVIOUSLY SPECIFIED BY CALLS TO    */
/*      THE APPROPRIATE PROCEDURES.                   */
/* VALUE: ERROR CODE                                  */
/**********************************************************/
    IF (A=[VDC$DRIVE] & 40H) !ZERO THEN /* READ ONLY DISK */
```

```
      DO;
        [ERROR]=(A=2); RETURN;
      END;
    CALL MAP;
    IF (A=[ERROR] \ A) !ZERO RETURN;
    STACK=HL; DE=(HL=[VDC$DMA]);
    HL=STACK; CALL [MOVBUF];
    [VDC$DRIVE]=(A=[VDC$DRIVE] \ 80H);   /* SET BIT 7 */
    END WRITE$FLOPPY;

EOF
```

```
/*******************************************************/
/************  TERMINAL INTERFACE MODULE  ***********/
/*******************************************************/
/*                                                     */
/*      TERMINAL INTERFACE MODULE PROVIDES THE MTS     */
/* INTERFACE WITH THE FOUR DISPLAY TERMINALS           */
/* ATTACHED TO THE SYCOR 440 SYSTEM.  THE MODULE IS    */
/* SEPARATED INTO FIVE BASIC SUBMODULES.  EACH         */
/* SUBMODULE CONTAINS THE MACROS NECESSARY FOR         */
/* LINKING WITHIN THE TERMINAL MODULE AND FOR          */
/* EXTERNAL MTS LINKAGE.                               */
/*                                                     */
/*  (1) DATA DECLARATIONS                              */
/*      THIS SUBMODULE PROVIDES ALL DECLARATIONS OF    */
/*      THE DATA STRUCTURES UTILIZED BY THE TERMINAL*/
/*      MODULE.                                        */
/*  (2) UTILITY PROCEDURES                             */
/*      THIS SUBMODULE CONTAINS THE BASIC UTILITY      */
/*      PROCEDURES WHICH PROVIDE COMMON REGISTER       */
/*      MANIPULATION AND PROCESSING REQUIRED BY        */
/*      MANY PROCEDURES THROUGHOUT THE REMAINDER       */
/*      OF THE TERMINAL INTERFACE MODULE.              */
/*          * COMPARE$PTRS        * GET$INDEX          */
/*          * GET$VALUE           * STORE$VALUE        */
/*          * CONVERT$NUMBR$TO$ASCII                   */
/*          * MOVE$BYTES          * SWAP$CURSOR        */
/*  (3) TERMINAL INTERFACE PRIMITIVES                  */
/*      THIS SUBMODULE CONTAINS THE PROCEDURES WHICH*/
/*      PROVIDE THE BASIC TERMINAL INTERFACE           */
/*      FUNCTIONS.                                     */
/*          * BLANK$DISPLAY       * GET$DISPLAY$ADDR   */
/*          * CHECK$CURSOR        * GET$TERM$STATUS    */
/*          * SCROLL$DISPLAY      * SEND$BEEP          */
/*          * SEND$CLICK          * UPDATE$CURSOR      */
/*          * GET$STATUS$ADDR                          */
/*  (4) KEY PROCESSING PROCEDURES                      */
/*      THIS SUBMODULE CONTAINS THE PROCEDURES WHICH*/
/*      PROVIDE THE BASIC TERMINAL INPUT PROCESSING.*/
/*      THEY ARE CALLED TO PROCESS EACH KEY ENTERED */
/*      AT A TERMINAL.  THEIR FUNCTIONS INCLUDE:       */
/*      CHECKING FOR AND CONVERTING LOWER TO UPPER     */
/*      CASE LETTERS IF REQUIRED; CHECKING FOR ANY     */
/*      KEY COMMANDS (WHICH INCLUDES ALL LINE          */
/*      EDITING); AND IF NOT A KEY COMMAND,            */
/*      DISPLAYING THE INPUT CHAR AT THE TERMINAL.     */
/*          * KEY$COMMAND     * TERM$INPUT$CNTL        */
/*  (5) TERMINAL INTERFACE SYSTEM FUNCTIONS            */
/*      THIS SUBMODULE CONTAINS THE PROCEDURES WHICH*/
/*      PROVIDE THE PROCESSING REQUIRED TO INTERFACE*/
/*      THE TERMINAL WITH THE REST OF THE MTS          */
/*      MODULES.  IT PROVIDES READING AND WRITING      */
/*      OF CHARACTERS FROM/TO THE TERMINALS;           */
/*      TERMINAL STATUS INFORMATION (E.G. WHETHER      */
/*      THERE'S INPUT AVAILABLE OR NOT); AND DISPLAY*/
/*      OF STATUS AND MTS MESSAGES ON THE TERMINAL     */
/*      STATUS LINE.                                   */
/*          * BLINK$CURSORS       * CLEAR$STATUS$LINE */
/*          * TERMINAL$STATUS     * READ$TERMINAL      */
/*          * WRITE$TERMINAL      * STATUS$MSG         */
/*          * MTS$MSG             * SIZE$MSG           */
/*******************************************************/

/*******************************************************/
```

```
/************  TERMINAL DISPLAY DESIGN  *************/
/*                                                   */
/*                                                   */
/*     GENERAL FORMAT OF EACH TERMINAL DISPLAY       */
/*                                                   */
/*                                                   */
/*     ----------------------------------------      */
/*     |0            STATUS LINE            63|      */
/*     ----------------------------------------      */
/*     |//////////////////////////////////////|      */
/*     ----------------------------------------      */
/*     |0      D                            63|      */
/*     ----------------------------------------      */
/*     |64         I                       127|      */
/*     ----------------------------------------      */
/*     |128    B       S                   191|      */
/*     ----------------------------------------      */
/*     |192        U       P               255|      */
/*     ----------------------------------------      */
/*     |256            F       L           319|      */
/*     ----------------------------------------      */
/*     |320                F       A       383|      */
/*     ----------------------------------------      */
/*     |384                    E       Y   447|      */
/*     ----------------------------------------      */
/*     |448                        R       511|      */
/*     ----------------------------------------      */
/*                                                   */
/*                                                   */
/*                   STATUS LINE                     */
/*     ----------------------------------------      */
/*     |0     VFD      39|40 MS 47|48 MSG 63|         */
/*     ----------------------------------------      */
/*     WHERE                                         */
/*    VFD - VIRTUAL FLOPPY DISK STATUS DISPLAY;      */
/*          CONTAINS THE INFORMATION ASSOCIATED      */
/*          WITH THE CURRENT TERMINAL USER'S         */
/*          VIRTUAL FLOPPY DISK DRIVE AND DISK       */
/*          NUMBER ASSIGNMENTS. (SEE STATUS$MSG      */
/*          PROC FOR DETAILS)                        */
/*     MS - DISPLAY OF THE MEMORY SIZE THE USER      */
/*          HAS REQUESTED. (SEE SIZE$MSG PROC)       */
/*    MSG - DISPLAY AREA FOR MTS MESSAGES WHICH      */
/*          ARE DISPLAYED IN RESPONSE TO A           */
/*          SYSTEM OR SERVICE CALL TO MTS.           */
/*          (SEE MTS$MSG PROC)                       */
/*                                                   */
/*                                                   */
/*                  DISPLAY BUFFER                   */
/*                  --------------                   */
/*                                                   */
/* THE DISPLAY BUFFER IS VIEWED AS A SINGLE          */
/* BUFFER FROM 0 TO 512 BYTES IN LENGTH.  THERE ARE */
/* FOUR DISPLAY BUFFER POINTERS WHICH PROVIDE THE    */
/* CONTROL OF INPUT FROM AND OUTPUT TO THE TERMINAL */
/* DISPLAY.  THESE POINTERS ARE:  CURRENT$LINE;      */
/* CURSOR; NEXT$CHAR; AND END$IBUFF.                 */
/* EACH POINTER UTILIZES TWO BYTES OF STORAGE TO     */
/* ACCOMMODATE A RANGE IN VALUE FROM 0 TO 512.       */
/* IN ADDITION TO THESE POINTERS, THERE IS A         */
/* TERMINAL STATUS BYTE, CALLED TERM$STATUS,         */
/* ASSOCIATED WITH EACH TERMINAL.  IT IS SET TO      */
/* ONE OF THREE VALUES:  INPUT$WAITING;              */
/* MTS$CMD$READY; IBUFF$EMPTY.  THESE ARE THE        */
/* PRIMARY DATA STRUCTURES PROVIDING TERMINAL I/O    */
/* CONTROL.                                          */
/*                                                   */
/* THE PRIMARY SYCOR HARDWARE CHARACTERISTIC WHICH   */
/* AFFECTED THE MTS TERMINAL INTERFACE WAS THE       */
/* RELATIVELY SLOW MINI-DISK ACCESS TIMES.  THIS HAS*/
/* A MAJOR IMPACT WHEN TRYING TO DESIGN AN           */
/* INTERACTIVE TIMESHARED SYSTEM.  IN ORDER TO       */
/* PROVIDE REASONABLE INTERACTIVE RESPONSE TIMES     */
```

```
/* TO USER ACTIONS, THE TERMINAL INTERFACE          */
/* MODULE PROVIDES ALL LINE EDITING FEATURES FOR    */
/* THE USER PRIOR TO TRANSFERING ANY DATA TO THE    */
/* USER'S PROGRAM. (SEE KEY$COMMAND PROC)           */
/* THE TERMINAL DISPLAY DESIGN UTILITZES TWO        */
/* SEPARATE BUFFERS TO PROVIDE THE USER WITH THE    */
/* CAPABILITY OF CONTINUING TO ENTER DATA PRIOR     */
/* TO THE USER'S PROGRAM BEING SWAPPED IN TO        */
/* PROCESS THE PREVIOUS INPUT LINE.                 */
/* THE FIRST BUFFER IS CALLED THE 'CURRENT LINE'    */
/* AND CONTAINS THE INPUT DATA WHICH IS CURRENTLY   */
/* BEING ENTERED BY THE USER.  THE CURRENT LINE CAN */
/* RANGE FROM 0 TO 512 BYTES IN LENGTH.  THIS IS THE*/
/* DATA THAT IS AFFECTED BY ANY LINE EDITING COMMAND*/
/* ENTERED BY THE USER.                             */
/* THE SECOND BUFFER IS THE INPUT LINE OR BUFFER.   */
/* THE CURRENT LINE BECOMES THE INPUT LINE WHENEVER */
/* A CARRIAGE RETURN OR ERROR RESET (MTS COMMAND    */
/* KEY) IS ENTERED.  THIS ACTION ALSO ESTABLISHES   */
/* A NEW CURRENT LINE.  THE INPUT LINE CONTAINS THE */
/* THE DATA WHICH IS TRANSFERRED TO THE USER PROGRAM*/
/* WHEN REQUESTED.  THUS THERE CAN BE AN INPUT LINE */
/* AND A CURRENT LINE ESTABLISHED AT ONE TIME.      */
/* THE CURRENT CURSOR POSITION ALWAYS SPECIFIES     */
/* WHERE THE NEXT CHARACTER WILL BE ENTERED, ON     */
/* INPUT, AND WHERE THE NEXT CHARACTER WILL BE      */
/* DISPLAYED DURING OUTPUT.                         */
/*                                                  */
/* THE FOLLOWING IS AN EXAMPLE OF POINTER           */
/* MANIPULATION DURING INPUT:                       */
/*  INITIALIZATION OR CLEAR SCREEN CMD:             */
/*      ALL POINTERS ARE SET TO ZERO AND            */
/*      TERM$STATUS = INPUT BUFFER EMPTY.           */
/*  USER ENTERS DATA - "SAMPLE INPUT DATA":         */
/*      CURRENT$LINE POINTS TO THE STARTING POSITION*/
/*      AND CURSOR ALWAYS POINTS TO THE NEXT        */
/*      POSITION TO FILL.  NOTE THAT AT THIS POINT  */
/*      ONLY CURSOR HAS BEEN MODIFIED. FOR THE      */
/*      INPUT DATA ABOVE IT WOULD BE POINTING TO    */
/*      DISPLAY BUFFER POSITION 17.                 */
/*  USER TERMINATES CURRENT LINE (I.E. ENTERS CR OR */
/*  MTS$CMD):                                       */
/*      END$IBUFF IS SET TO CURRENT CURSOR POSITION.*/
/*      CURSOR IS SET TO LEFT MOST POSITION OF NEXT */
/*          LINE ON DISPLAY.                        */
/*      CURRENT$LINE IS SET TO NEW CURSOR POSITION. */
/*      TERM$STATUS IS SET TO INPUT WAITING.        */
/*                                                  */
/* THE RESULTING POINTER POSITIONS ARE SHOWN FOR THE*/
/* SAMPLE INPUT DATA AND CR CHARACTERS ENTERED.     */
/* (WHERE * = CURRENT CURSOR POSITION)              */
/*                                                  */
/*      NC                 EIB                      */
/*      ----------------------------------------    */
/*      |SAMPLE INPUT DATA                     |    */
/*      ----------------------------------------    */
/*      |*                                     |    */
/*      ----------------------------------------    */
/*      CL                                          */
/*                                                 .*/
/* THE SAMPLE INPUT DATA IS NOW AVAILABLE FOR THE   */
/* USER'S PROGRAM WHEN IT'S TIMESLICE COMES UP.     */
/* THE NEXT$CHAR (NC) POINTER SPECIFIES THE NEXT    */
/* CHARACTER TO BE READ AND RETURNED TO THE USER    */
/* PROGRAM.  WHEN NEXT$CHAR = END$IBUFF, A CARRIAGE */
/* RETURN (CR) IS RETURNED TO THE CALLING USER      */
/* PROGRAM.                                         */
/* THERE ARE THREE OCCASIONS WHEN THE NEXT$CHAR     */
/* POINTER IS RESET EQUAL TO THE CURRENT$LINE       */
/* POINTER:                                         */
/*  (1) FOR A CLEAR SCREEN COMMAND.                 */
/*  (2) WHEN READ$TERMINAL PROC DETECTS THAT THE    */
/*      END OF THE INPUT BUFFER HAS BEEN REACHED.   */
```

```
/*  (3) WHEN WRITE$TERMINAL PROC OUTPUTS CHARACTERS */
/*      TO THE TERMINAL FROM THE USER'S PROGRAM.    */
/*                                                  */
/* THE OUTPUT OF DATA FROM THE USER'S PROGRAM TO    */
/* THE TERMINAL RESULTS IN THE FOLLOWING:           */
/*  (1) THE CHARACTER IS DISPLAYED AT THE CURRENT   */
/*      CURSOR POSITION.                            */
/*  (2) THE CURSOR POSITION IS INCREMENTED.         */
/*  (3) THE CURRENT$LINE AND NEXT$CHAR POINTERS ARE */
/*      SET EQUAL TO THE NEW CURSOR POSITION.       */
/*  (4) THE TERMINAL STATUS IS SET TO EMPTY.        */
/*                                                  */
/*                                                  */
/* ANOTHER DESIGN CONSIDERATION WAS THE REQUIREMENT */
/* FOR THE TERMINAL MODULE TO PROVIDE A BLINKING    */
/* CURSOR DISPLAY AT EACH TERMINAL.  THIS REQUIRED  */
/* SPECIAL PROCESSING TO ENSURE THAT THE CURSOR     */
/* CHARACTER (05FH) DID NOT GET LOST DURING THE     */
/* CURSOR UPDATE AND MANIPULATION FUNCTIONS         */
/* ACCOMPLISHED BY THE KEY PROCESSING AND SYSTEM    */
/* FUNCTION SUBMODULES. CHECK$CURSOR PROC(PRIMITIVE */
/* SUBMODULE) PROVIDES THIS FUNCTION.               */
/*                                                  */
/*                                                  */
/**************************************************/
/**************************************************/


/**************************************************/
/*****  TERMINAL INTERFACE DATA DECLARATIONS *******/
/**************************************************/


[MACRO IBUFF$EMPTY  '0'  ]
[MACRO CURSOR$CHAR  '5FH']

/**************************************************/
/*********  TERMINAL INTERFACE DECLARATIONS ********/

  /* ASCII - CONTAINS DATA FOR MATRIX CODE TO ASCII*/
  /*         CONVERSION.                           */

DECLARE ASCII DATA (1EH,1CH,1BH,5DH,5BH,29H,28H,7FH,
    26H,3DH,25H,24H,23H,40H,21H,2AH,0AH,0CH,0BH,0A0H,
    0DH,50H,4FH,15H,55H,59H,54H,52H,45H,57H,51H,49H,
    0DH,9,31H,9,22H,3AH,4CH,0A5H,4AH,48H,47H,46H,44H,53H,
    41H,4BH,0FFH,30H,20H,0A3H,0A2H,3FH,3EH,3CH,4DH,4EH,
    42H,56H,43H,58H,5AH,0FFH,
    0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,7FH,0FFH,0FFH,
    0FFH,0A4H,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0A0H,
    0DH,0FFH,0FFH,15H,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,
    0FFH,0FFH,0FFH,0FFH,0FFH,09,0FFH,0FFH,0FFH,0A5H,
    0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,
    20H,0A3H,0A2H,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,
    0A1H,0FFH,0FFH,0FFH,
    0FFH,0FFH,0FFH,5FH,0FFH,7DH,7BH,7FH,0FFH,7EH,5CH,
    0A4H,60H,5EH,7CH,0FFH,0FFH,0FFH,0FFH,0A0H,0DH,
    10H,0FH,15H,15H,19H,14H,12H,05,17H,11H,09,60H,
    5EH,7CH,09,0FFH,0FFH,0CH,0A5H,0AH,08,07,06,04,
    13H,01,0BH,0FFH,7DH,20H,0A3H,0A2H,0FFH,0FFH,0FFH,
    0DH,0EH,02,16H,03,18H,1AH,0FFH,
    39H,38H,37H,2DH,2BH,30H,39H,7FH,37H,36H,35H,34H,
    33H,32H,31H,38H,36H,35H,34H,0A0H,0DH,70H,6FH,15H,
    75H,79H,74H,72H,65H,77H,71H,69H,33H,32H,31H,09,
    27H,3BH,6CH,0A5H,6AH,68H,67H,66H,64H,73H,61H,6BH,
    0FFH,30H,20H,0A3H,0A2H,2FH,2EH,2CH,6DH,6EH,62H,
    76H,63H,78H,7AH,0FFH);

/**************************************************/
/* STATUS$BASE  - START OF STATUS LINE AT EACH TERM */
/* DISPLAY$BASE - START OF DISPLAY LINES           */
/**************************************************/
DECLARE STATUS$BASE  DATA
```

```
                      (00H,07H,40H,09H,80H,0BH,0C0H,0DH);
DECLARE DISPLAY$BASE DATA
                      (40H,07H,80H,09H,0C0H,0BH,00H,0EH);

/*******************************************************/
/* MTS$MESSAGE - DATA VECTOR CONTAINING ALL THE MTS */
/*              MESSAGES WHICH MAY BE DISPLAYED IN */
/*              THE MTS MSG FIELD OF THE STATUS    */
/*              LINE.                               */
/* SIZE$MESSAGE- DATA VECTOR CONTAINING THE TEXT     */
/*              PORTION OF THE SIZE MSG FIELD OF    */
/*              THE STATUS LINE.                    */
/*******************************************************/
DECLARE MTS$MESSAGE DATA ('                ',
'   INVALID CMD  ',' DISK NOT AVAIL','   DISK IN USE  ',
'   DISK NR ERROR','     KEY ERROR   ',' DRIVE LTR ERROR',
' PRINTER NOT RDY',' HARDWARE ERROR ','   TASK DELETED  ',
' DRIVE NOT AVAIL');

DECLARE SIZE$MESSAGE DATA ('K MTS ');

/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
/* THE NEXT FOUR DECLARATIONS PROVIDE THE POINTERS      */
/* UTILIZED TO CONTROL THE INPUT/OUTPUT AT EACH         */
/* TERMINAL.                                            */
/* CURSOR - SPECIFIES THE CURRENT ADDRESS WHERE THE     */
/*          CURSOR  IS TO BE DISPLAYED.                 */
/* CURRENT$LINE - ADDRESS WHICH POINTS TO INITIAL BYTE */
/*          OF CURRENT USER INPUT LINE. THIS LINE HAS  */
/*          NOT YET RECEIVED A 'CR' AND THUS IS NOT YET*/
/*          CONSIDERED AN INPUT BUFFER.                 */
/* NEXT$CHAR - POINTS TO NEXT CHAR TO BE PROCESSED FROM*/
/*          THE INPUT BUFFER. AN INPUT IS DEFINED AS A */
/*          STRING OF ASCII CHARACTERS (FROM 1 TO 512) */
/*          WHICH HAS BEEN TERMINATED BY A 'CR' OR      */
/*          'MTS$CMD' KEY BY THE USER.                  */
/* END$IBUFF - POINTS TO BYTE POSITION IN INPUT BUFFER */
/*          WHERE 'CR' OR 'MTS$CMD' WAS RECEIVED.       */
/*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */

DECLARE CURSOR       (8) BYTE INITIAL(0,0,0,0,0,0,0,0);
DECLARE CURRENT$LINE (8) BYTE INITIAL(0,0,0,0,0,0,0,0);
DECLARE NEXT$CHAR    (8) BYTE INITIAL(0,0,0,0,0,0,0,0);
DECLARE END$IBUFF    (8) BYTE INITIAL(0,0,0,0,0,0,0,0);

/* CAPITALIZE - SET TO ONE IF TERMINAL IN CAP MODE */

DECLARE CAPITALIZE (4) BYTE INITIAL (1,1,1,1);

/* TERM$STATUS - CONTAINS THE CURRENT STATUS OF      */
/*              EACH TERMINAL'S INPUT BUFFER,        */
/*              EITHER INPUT WAITING;                */
/*              MTS CMD READY; OR IBUFF EMPTY.       */

DECLARE TERM$STATUS (4) BYTE INITIAL ([IBUFF$EMPTY],
    [IBUFF$EMPTY],[IBUFF$EMPTY],[IBUFF$EMPTY]);

/* SWAP$POS - FOR EACH TERMINAL THERE IS A ONE BYTE */
/*            SAVE AREA WHICH IS USED DURING CURSOR */
/*            BLINKING PROCESSING                   */

DECLARE SWAP$POS (4) BYTE INITIAL ([CURSOR$CHAR],
    [CURSOR$CHAR],[CURSOR$CHAR],[CURSOR$CHAR]);

EOF
/*******************************************************/
/***  TERMINAL  INTERFACE UTILITY PROCEDURES  *******/
/*******************************************************/


[MACRO TRUE          '0FFH']
[MACRO FALSE         '0'   ]
[MACRO SWAP$POS      '11FDH']
```

```
COMPARE$PTRS: PROCEDURE;
/*******************************************************/
/* COMPARES TWO POINTERS (2 BYTES EACH) TO DETERMINE*/
/* IF THEY ARE EQUAL.                                  */
/* INPUT:   DE - ADDRESS OF FIRST PTR                  */
/*          HL - ADDRESS OF SECOND PTR                 */
/* OUTPUT:   A - TRUE IF EQUAL, FALSE OTHERWISE.       */
/* CALLED BY:   KEY$COMMAND; READ$TERMINAL;            */
/*******************************************************/
    CY=0;
    IF (A=M(DE)--M(HL)) !ZERO THEN
        A=[FALSE]
    ELSE
        DO;
        DE=DE+1; HL=HL+1;
        IF (A=M(DE)--M(HL)) !ZERO THEN
            A=[FALSE]
        ELSE
            A=[TRUE];
        END;
    END COMPARE$PTRS;


CONVERT$NUMBR$TO$ASCII: PROCEDURE;
/*******************************************************/
/* CONVERTS THE SPECIFIED NUMBER TO A DISPLAYABLE     */
/* TWO DIGIT DECIMAL NUMBER (MAX VALUE IS 99).        */
/* INPUT:  A - NUMBER TO BE CONVERTED.                */
/* OUTPUT: B - LEFT MOST DIGIT TO BE DISPLAYED.       */
/*         C - RIGHT MOST DIGIT TO BE DISPLAYED.      */
/* CALLED BY:  SIZE$MSG; STATUS$MSG;                  */
/*******************************************************/
    B=0; C=A;
    DO WHILE (A::10) PLUS;
        B=B+1;
        C=(A=C-10);
        END;
    C=(A=A+30H);
    B=(A=B+30H);
    END CONVERT$NUMBR$TO$ASCII;


GET$INDEX: PROCEDURE;
/*******************************************************/
/* USED TO GET THE INDEX INTO AN ADDRESS ARRAY BY     */
/* COMPUTING THE OFFSET FROM A GIVEN BASE ADDRESS.    */
/* INPUT:   A - OFFSET VALUE (NORMALLY THE TASK OR    */
/*              TERMINAL NUMBER).                     */
/*         HL - BASE ADDRESS                          */
/* OUTPUT: DE- ARRAY OFFSET                           */
/*         HL- ARRAY OFFSET (HL=DE)                   */
/*         BC- COMPUTED OFFSET                        */
/* CALLED BY:  SCROLL$DISPLAY; UPDATE$CURSOR;         */
/*             KEY$COMMAND; TERM$INPUT$CNTL;          */
/*             READ$TERMINAL; WRITE$TERMINAL          */
/*******************************************************/
    CY=0; B=0;
    C=(A=<<A);  /* SET ADDRESS OFFSET TO OFFSET*2     */
    HL=HL+BC; DE=HL;
    END GET$INDEX;


GET$VALUE: PROCEDURE;
/*******************************************************/
/* GETS A 2 BYTE VALUE FROM MEMORY.                   */
/* INPUT: DE - ADDRESS OF 2 BYTE VECTOR; THE          */
/*             CONTENTS ARE TO BE STORED IN THE       */
/*             HL REGISTER.                           */
/* OUTPUT: HL - CONTENTS OF 2 BYTE VECTOR             */
/*         DE - ADDRESS OF THE HIGH ORDER BYTE        */
/* CALLED BY:   SCROLL$DISPLAY; UPDATE$CURSOR;        */
```

```
/*                KEY$COMMAND; READ$TERMINAL;            */
/*                WRITE$TERMINAL; GET$DISPLAY$ADDR       */
/*******************************************************/
    L=(A=M(DE)); DE=DE+1; H=(A=M(DE));
    END GET$VALUE;


STORE$VALUE: PROCEDURE;
/*******************************************************/
/* STORE A 2 BYTE VALUE INTO MEMORY.                     */
/* INPUT: HL - VALUE TO BE STORED INTO MEMORY            */
/*        DE - ADDRESS OF HIGH ORDER BYTE                */
/* CALLED BY:  UPDATE$CURSOR; KEY$COMMAND;               */
/*             READ$TERMINAL; WRITE$TERMINAL;            */
/*              GET$DISPLAY$ADDR;                        */
/*******************************************************/
    M(DE)=(A=H); DE=DE-1; M(DE)=(A=L);
    END STORE$VALUE;


MOVE$BYTES: PROCEDURE;
/*******************************************************/
/* MOVES BYTES OF DATA FROM ONE MEMORY LOCATION TO      */
/* ANOTHER.                                              */
/* INPUT: BC - NUMBER OF BYTE TO BE MOVED.               */
/*        DE - STARTING MEMORY ADDRESS TO MOVE           */
/*             BYTES TO (DESTINATION).                   */
/*        HL - STARTING MEMORY ADDRESS TO MOVE           */
/*             BYTES FROM (SOURCE).                      */
/* CALLED BY:  SIZE$MSG; MTS$MSG;                        */
/*******************************************************/
    REPEAT;
        M(DE)=(A=M(HL));
        DE=DE+1; HL=HL+1;
        BC=BC-1; A=0;
    UNTIL (A::B) ZERO & (A::C) ZERO;
    END MOVE$BYTES;


SWAP$CURSOR: PROCEDURE;
/*******************************************************/
/* SWAP THE SPECIFIED TERMINAL'S CURRENT CURSOR          */
/* POSITION CHAR WITH THE SWAP$POS CHAR.                 */
/* INPUT: A  - TERMINAL NUMBER                           */
/*        HL - DISPLAY ADDRESS OF CURSOR POSITION        */
/* CALLED BY:  BLINK$CURSORS; CHECK$CURSOR;              */
/*******************************************************/
    DE=HL;              /* SAVE DISPLAY ADDRESS          */
    B=0; C=A;
    HL=[SWAP$POS]+BC;     /* GET SWAP ADDRESS            */
    B=(A=M(DE));          /*  SWAP                       */
    M(DE)=(A=M(HL));
    M(HL)=(A=B);
    END SWAP$CURSOR;


EOF
/*******************************************************/
/********* TERMINAL INTERFACE PRIMITIVES  *************/
/*******************************************************/


[MACRO TRUE                 '0FFH']
[MACRO FALSE                '0'   ]
[MACRO BLANK                '20H' ]
[MACRO IBUFF$EMPTY          '0'   ]
[MACRO DISPLAY$SIZE         '512' ]
[MACRO TERM$PORT            '03FH']
[MACRO CURSOR$CHAR          '05FH']

/*******************************************************/
/************  INTERNAL LINKAGE MACROS  ***************/
```

```
[INT TB]                [TB := 1000H]

[MACRO STATUS$BASE   '[HEX TB + 0106H]']
[MACRO DISPLAY$BASE  '[HEX TB + 0111H]']
[MACRO CURSOR        '[HEX TB + 01D5H]']
[MACRO CURRENT$LINE  '[HEX TB + 01DDH]']
[MACRO NEXT$CHAR     '[HEX TB + 01E5H]']
[MACRO END$IBUFF     '[HEX TB + 01EDH]']
[MACRO TERM$STATUS   '[HEX TB + 01F9H]']

[MACRO GET$INDEX     '[HEX TB + 024CH]']
[MACRO GET$VALUE     '[HEX TB + 0259H]']
[MACRO STORE$VALUE   '[HEX TB + 0262H]']
[MACRO SWAP$CURSOR   '[HEX TB + 027EH]']


BLANK$DISPLAY: PROCEDURE;
/*****************************************************/
/* PLACES BLANKS INTO THE INDICATED AREA OF A        */
/* TERMINAL DISPLAY.                                 */
/* INPUT:  BC - STARTING ADDRESS (RANGE 0 TO 511)    */
/*         DE - NUMBER OF BYTES TO SET TO BLANK      */
/*         HL - BASE ADDRESS                         */
/* CALLED BY:  KEY$COMMAND; SCROLL$DISPLAY;          */
/*             CLEAR$STATUS$LINE; MTS$MSG;           */
/*****************************************************/
    HL=HL+BC;
    REPEAT;
        M(HL)=(A= [BLANK]);
        HL=HL+1;
        DE=DE-1; A=0;
    UNTIL (A::D) ZERO  & (A::E) ZERO;
    END BLANK$DISPLAY;


GET$DISPLAY$ADDR: PROCEDURE;
/*****************************************************/
/* GETS A MEMORY ADDRESS IN THE DISPLAY BUFFER USING*/
/* THE DISPLAY BUFFER PTR AS OFFSET FROM THE DISPLAY*/
/* BASE ADDRESS.                                     */
/* INPUT:  BC - TERMINAL NUMBER OFFSET.              */
/*         DE - ADDRESS OF DISPLAY BUFFER PTR        */
/* OUTPUT: HL - MEMORY ADDRESS IN DISPLAY BUFFER.    */
/*         BC - DISPLAY PTR VALUE.                   */
/* CALLED BY:  TERM$INPUT$CNTL; KEY$COMMAND;         */
/*             BLINK$CURSORS; READ$TERMINAL;         */
/*             WRITE$TERMINAL;                       */
/*****************************************************/
    DECLARE PTR(2) BYTE;
    CALL [GET$VALUE];
    PTR=HL;            /* SAVE DISPLAY PTR VALUE     */
    DE=(HL=[DISPLAY$BASE]+BC);/*GET$VALUE PARAMETER */
    BC=(HL=PTR);      /* GET DISPLAY PTR VALUE       */
    CALL [GET$VALUE]; /* GET DISPLAY BASE            */
    HL=HL+BC;         /* GET DISPLAY ADDRESS         */
    END GET$DISPLAY$ADDR;


CHECK$CURSOR: PROCEDURE;
/*****************************************************/
/* PRIOR TO CHANGING THE CURRENT CURSOR POSITION OR */
/* DISPLAYING A CHARACTER AT THE CURRENT CURSOR POS,*/
/* A CHECK IS ALWAYS MADE TO ENSURE THAT THE         */
/* CURRENT DISPLAY IS A DATA CHARACTER AND NOT THE   */
/* CURSOR ITSELF (I.E. 5FH).  IF IT IS THE CURSOR    */
/* A SWAP IS MADE.                                   */
/* INPUT: A - TERMINAL NUMBER                        */
/* CALLED BY: KEY$COMMAND; TERM$INPUT$CNTL;          */
/*            WRITE$TERMINAL;                        */
/*****************************************************/
    DECLARE T BYTE;
    T=A; HL=[CURSOR];
    CALL [GET$INDEX];
```

```
    CALL GET$DISPLAY$ADDR;
    IF (A=M(HL); A::[CURSOR$CHAR]) ZERO THEN
        DO;
        A=T;
        CALL [SWAP$CURSOR];
        END;
    END CHECK$CURSOR;


GET$STATUS$ADDR: PROCEDURE;
/*****************************************************/
/* GETS THE BASE ADDRESS OF THE STATUS LINE FOR THE */
/* SPECIFIED TERMINAL.                               */
/* INPUT:  A  - TERMINAL NUMBER                      */
/* OUTPUT: HL - MEMORY ADDRESS OF FIRST BYTE IN      */
/*              STATUS LINE.                         */
/* CALLED BY:   CLEAR$STATUS$LINE; MTS$MSG;          */
/*              SIZE$MSG; STATUS$MSG;                */
/*****************************************************/
    HL=[STATUS$BASE];
    CALL [GET$INDEX];
    CALL [GET$VALUE];
    END GET$STATUS$ADDR;


GET$TERM$STATUS: PROCEDURE;
/*****************************************************/
/* RETRIEVES THE TERMINAL STATUS FOR THE INDICATED  */
/* TERMINAL. TERMINAL STATUS SPECIFIES WHETHER       */
/* OR NOT THERE IS AN INPUT BUFFER OR MTS COMMAND    */
/* READY FOR PROCESSING FOR THAT TERMINAL.           */
/* INPUT:  A - TERMINAL NUMBER                       */
/* OUTPUT: A - TERMINAL STATUS                       */
/* CALLED BY:  KEY$COMMAND; TERMINAL$STATUS;         */
/*             UPDATE$CURSOR; MONITOR (MON MOD);     */
/*****************************************************/
    B=0; C=A;
    HL=[TERM$STATUS]+BC;
    A=M(HL);
    END GET$TERM$STATUS;


SCROLL$DISPLAY: PROCEDURE;
/*****************************************************/
/* SCROLLS THE DISPLAY FOR THE INDICATED TERMINAL.  */
/* INPUT:  A - TERMINAL NUMBER                       */
/* CALLED BY:  UPDATE$CURSOR                         */
/*****************************************************/
    DECLARE TERM BYTE;
    DECLARE POS(2) BYTE;
    TERM=A; HL=[DISPLAY$BASE];
    CALL [GET$INDEX];
    CALL [GET$VALUE]; DE=HL; /*DE=DISPLAY BASE ADDR */
    POS=HL;               /* SAVE DISPLAY BASE ADDRESS*/
    BC=64;
    BC=(HL=HL+BC);        /* BC=DISPLAY BASE + 64     */
    HL=448;
    REPEAT;
        M(DE)=(A=M(BC));
        BC=BC+1; DE=DE+1;
        HL=HL-1; A=0;
    UNTIL (A::H) ZERO  & (A::L) ZERO;
    BC=448;               /* SETUP PARAMETERS FOR     */
    DE=64; HL=POS;        /*  BLANK$DISPLAY PROC      */
    CALL BLANK$DISPLAY;
    END SCROLL$DISPLAY;


SEND$BEEP: PROCEDURE;
/*****************************************************/
/* SENDS A BEEP TO THE INDICATED TERMINAL.          */
/* THE FORM OF THE TERMINAL ALERT CONTROL BYTE IS:  */
/*         7  6  5  4  3  2  1  0                    */
```

```
/*          -------------------------                    */
/*          |A3|C3|A2|C2|A1|C1|A0|C0|                    */
/*          -------------------------                    */
/* WHERE:                                                */
/*          A(I) = 1; GENERATES AN ALARM AT STATION I.   */
/*          C(I) = 1; GENERATES A CLICK AT STATION I.    */
/* INPUT:   A - TERMINAL NUMBER                          */
/* CALLED BY:   KEY$COMMAND;                             */
/*********************************************************/
     H=0; L=A;
     IF (A::0) PLUS  8 (A::4) MINUS THEN
          DO CASE HL;
          OUT([TERM$PORT])=(A=2);
          OUT([TERM$PORT])=(A=8);
          OUT([TERM$PORT])=(A=20H);
          OUT([TERM$PORT])=(A=80H);
          END;
     END SEND$BEEP;


SEND$CLICK: PROCEDURE;
/*********************************************************/
/* SENDS A CLICK TO THE INDICATED TERMINAL. SEE          */
/* SEND$BEEP PROC FOR DEFINITION OF TERMINAL ALERT       */
/* CONTROL BYTE.                                         */
/* INPUT:   A - TERMINAL NUMBER                          */
/* CALLED BY:   UPDATE$CURSOR                            */
/*********************************************************/
     H=0; L=A;
     IF (A::0) PLUS  8 (A::4) MINUS THEN
          DO CASE HL;
          OUT([TERM$PORT])=(A=1);
          OUT([TERM$PORT])=(A=4);
          OUT([TERM$PORT])=(A=10H);
          OUT([TERM$PORT])=(A=40H);
          END;
     END SEND$CLICK;


UPDATE$CURSOR: PROCEDURE;
/*********************************************************/
/* CONTROLS THE UPDATING OF THE CURSOR POSITION. THE*/
/* PRIMARY CONCERN IS TO CHECK FOR SCROLLING PRIOR       */
/* TO UPDATING THE CURSOR. SCROLLING IS NOT ALLOWED      */
/* IF SCROLLING WILL DESTROY ANY INPUT DATA NOT YET      */
/* PROCESSED.                                            */
/* SUBPROCEDURES: CHECK$SCROLL$LOCKOUT                   */
/*                UPDATE$DISPLAY$PTRS                    */
/* INPUT:     A   - TERMINAL NUMBER                      */
/*            HL  - VALUE TO WHICH CURSOR POSITION IS    */
/*                  TO BE SET.                           */
/* CALLED BY:       KEY$COMMAND; TERM$INPUT$CNTL;        */
/*                  WRITE$TERMINAL;                      */
/*********************************************************/
     DECLARE T BYTE;
     DECLARE POS(2) BYTE;

     CHECK$SCROLL$LOCKOUT: PROCEDURE;
     /****************************************************/
     /* CHECKS TO SEE IF THERE IS AN INPUT BUFFER        */
     /* READY. IF SO, CHECKS TO SEE IF SCROLLING WILL*/
     /* DESTROY INPUT BUFFER. IF SO, RETURN TRUE,        */
     /* ELSE RETURN FALSE.                               */
     /* OUTPUT: A - TRUE IF SCROLLING IS LOCKED OUT      */
     /****************************************************/
          A=T;
          CALL GET$TERM$STATUS;
          IF (A=A-[IBUFF$EMPTY]) !ZERO THEN
               DO;      /* CHECK NEXT$CHAR PTR           */
               A=T; HL=[NEXT$CHAR];
               CALL [GET$INDEX];/* GET NEXT$CHAR OFFSET*/
               CALL [GET$VALUE]; /* GET NEXT$CHAR VALUE*/
               BC= -64; CY=0;
```

```
        IF (HL=HL+BC) CY THEN /* SCROLL OK          */
            A=[FALSE]
        ELSE        /*  SCROLL LOCKEDOUT             */
            A=[TRUE];
        END
    ELSE        /* NO INPUT BUFFER, SCROLL OK    */
        A=[FALSE];
    END CHECK$SCROLL$LOCKOUT;


UPDATE$DISPLAY$PTRS: PROCEDURE;
/************************************************/
/* UPDATES ALL DISPLAY PTRS TO REFLECT THE      */
/* SCROLLING OF THE DISPLAY. USES SET$PTR TO    */
/* DECREMENT AND STORE THE POINTER VALUES.      */
/* SUBPROCEDURE: SET$PTR;                       */
/************************************************/

    SET$PTR: PROCEDURE;
    /********************************************/
    /* SETS THE SPECIFIED PTR TO PTR-64, AND    */
    /* STORES THE RESULT.                       */
    /* INPUT: DE - ADDRESS OF PTR               */
    /********************************************/
        CALL [GET$VALUE];
        BC=-64; HL=HL+BC;
        CALL [STORE$VALUE];
        END SET$PTR;

    /********************************************/
    /* START OF UPDATE$DISPLAY$PTRS PROCESSING  */
    /********************************************/

        /* SET CURSOR TO LEFT MARGIN OF 8TH LINE*/
        /* ON DISPLAY.                          */
    A=T; HL=[CURSOR];
    CALL [GET$INDEX];
    HL=448; DE=DE+1;
    CALL [STORE$VALUE];
        /* SET NEXT$CHAR = NEXT$CHAR - 64       */
    DE=(HL=[NEXT$CHAR]+BC);
    CALL SET$PTR;
        /* SET CURRENT$LINE = CURRENT$LINE - 64 */
    A=T; HL=[CURRENT$LINE];
    CALL [GET$INDEX];
    CALL SET$PTR;
        /* SET END$IBUFF = END$IBUFF - 64       */
    A=T; HL=[END$IBUFF];
    CALL [GET$INDEX];
    CALL SET$PTR;
    END UPDATE$DISPLAY$PTRS;

/************************************************/
/* START OF UPDATE$CURSOR PROCESSING            */
/************************************************/

T=A;        /* SAVE INPUT TERMINAL NUMBER       */
POS=HL;     /* SAVE INPUT CURSOR POSITION       */
BC= -[DISPLAY$SIZE]; CY=0;
IF (HL=HL+BC) CY THEN   /*  SCROLLING REQUIRED  */
    DO;
    CALL CHECK$SCROLL$LOCKOUT;
    IF (A=>>A) CY THEN  /* SCROLLING LOCKED OUT */
        DO;
        A=T;
        CALL SEND$CLICK;
        END
    ELSE                /* SCROLLING IS OK      */
        DO;
        A=T;
        CALL SCROLL$DISPLAY;
        CALL UPDATE$DISPLAY$PTRS;
        END;
```

```
        END
    ELSE    /* NO SCROLLING REQUIRED; UPDATE CURSOR */
        DO;
        A=T; HL=[CURSOR];
        CALL [GET$INDEX];    /* GET CURSOR PTR ADDR  */
        HL=POS; /* SETUP [STORE$VALUE] PARAMETERS    */
        DE=DE+1;
        CALL [STORE$VALUE]; /* UPDATE CURSOR PTR     */
        END;
    END UPDATE$CURSOR;


EOF
/*****************************************************/
/******  TERMINAL KEY PROCESSING PROCEDURES  ********/
/*****************************************************/


[MACRO TRUE                         '0FFH']
[MACRO FALSE                        '0'   ]
[MACRO BLANK                        '20H' ]
[MACRO CURSOR$CHAR                  '5FH' ]
[MACRO INPUT$WAITING                '0FFH']
[MACRO MTS$CMD$READY                '0F0H']
[MACRO IBUFF$EMPTY                  '0'   ]


/*****************************************************/
/**********  MTS KEY COMMAND MACROS  ****************/

[MACRO MTS$CMD                      '0A0H']
[MACRO CR                           '0DH' ]
[MACRO CHAR$DELETE                  '07FH']
[MACRO LINE$DELETE                  '015H']
[MACRO CAPITAL                      '0A1H']
[MACRO CURSOR$LEFT                  '0A2H']
[MACRO CURSOR$RIGHT                 '0A3H']
[MACRO CLEAR$SCREEN                 '0A4H']
[MACRO ENTER                        '0A5H']
[MACRO RST$CMD                      '01EH']


/*****************************************************/
/*****  INTERNAL AND EXTERNAL LINKAGE MACROS  *******/

[INT GB TB]    [GB := 0]    [TB := 1000H]

[MACRO TCT$STATUS          '[HEX GB + 3E85H]']

[MACRO ASCII               '[HEX TB + 0003H]']
[MACRO DISPLAY$BASE        '[HEX TB + 0111H]']
[MACRO CURSOR              '[HEX TB + 01D5H]']
[MACRO CURRENT$LINE        '[HEX TB + 01DDH]']
[MACRO NEXT$CHAR           '[HEX TB + 01E5H]']
[MACRO END$IBUFF           '[HEX TB + 01EDH]']
[MACRO CAPITALIZE          '[HEX TB + 01F5H]']
[MACRO SWAP$POS            '[HEX TB + 01FDH]']

[MACRO COMPARE$PTRS        '[HEX TB + 0213H]']
[MACRO GET$INDEX           '[HEX TB + 024CH]']
[MACRO GET$VALUE           '[HEX TB + 0259H]']
[MACRO STORE$VALUE         '[HEX TB + 0262H]']

[MACRO BLANK$DISPLAY       '[HEX TB + 02A3H]']
[MACRO GET$DISPLAY$ADDR    '[HEX TB + 02B7H]']
[MACRO CHECK$CURSOR        '[HEX TB + 02D0H]']
[MACRO GET$TERM$STATUS     '[HEX TB + 02F9H]']
[MACRO SEND$BEEP           '[HEX TB + 033EH]']
[MACRO UPDATE$CURSOR       '[HEX TB + 03C2H]']


KEY$COMMAND: PROCEDURE;
/*****************************************************/
```

```
/* CHECKS FOR A TERMINAL KEY COMMAND FOR EVERY KEY   */
/* INTERRUPT RECEIVED.                               */
/* KEY COMMANDS ARE:                                 */
/*      CMD          KEY          RESULT             */
/*      ---          ---          ------             */
/*  MTS CMD        ERROR RESET   SENDS A COMMAND TO  */
/*                               MTS FOR PROCESSING. */
/*  CR             NEW LINE;     TERMINATES THE      */
/*                 ENTER; SHIFT  CURRENT LINE AND    */
/*                 CR; I/O CTL M;ESTABLISHES IT AS   */
/*                               THE CURRENT INPUT   */
/*                               BUFFER.             */
/* CHAR DELETE     BACK SPACE    DELETES THE LAST    */
/*                               CHAR ENTERED.       */
/* LINE DELETE     NEXT FMAT     DELETES THE CURRENT */
/*                               LINE.               */
/* CAPTIALIZE      FS  C         FLIP/FLOP USED TO   */
/*                               SET OR CLEAR THE    */
/*                               TERMINAL INPUT MODE */
/*                               TO UPPER OR LOWER   */
/*                               CASE LETTERS.       */
/* CLEAR SCREEN      FS  $       CLEARS THE 512 CHAR */
/*                               DISPLAY BUFFER.     */
/* CURSOR LEFT       <--         MOVES CURSOR POS    */
/*                               ONE POSITION TO THE */
/*                               LEFT.               */
/* CURSOR RIGHT      -->         MOVES CURSOR POS    */
/*                               ONE POSITION TO THE */
/*                               RIGHT.              */
/* RST CMD           SHIFT RS    EXECUTE A RST 7     */
/*                               INSTRUCTION. FOR    */
/*                               USE WITH CP/M DDT.  */
/* SUBPROCEDURES: ACCEPT$INPUT; CHECK$LEFT$MARGIN;   */
/*                DELETE$CHAR; CLEAR$PTRS; MTS$CMD;  */
/*                TERMINATE$CL; CARRIAGE$RETURN;     */
/*                CHAR$DELETE; LINE$DELETE; CAPITAL;*/
/*                CLEAR$SCREEN; CURSOR$LEFT;         */
/*                CURSOR$RIGHT; CHECK$CASE; RST$CMD;*/
/* INPUT:    A   - TERMINAL NUMBER                   */
/*           C   - ASCII CHAR RECEIVED               */
/* OUTPUT:   A   - TRUE IF CHAR = KEY COMMAND        */
/* CALLED BY:      TERM$INPUT$CNTL;                  */
/*****************************************************/
    DECLARE (CHAR,T) BYTE;
    DECLARE RESPONSE BYTE;

    ACCEPT$INPUT: PROCEDURE;
    /*************************************************/
    /* CHECKS THE TERMINAL'S CURRENT STATUS TO       */
    /* DETERMINE IF THIS NEW INPUT BUFFER SHOULD BE */
    /* ACCEPTED.  IF SO, RETURNS TRUE, ELSE FALSE.  */
    /* OUTPUT: A  - TRUE IS INPUT CAN BE ACCEPTED.  */
    /*         HL - IF A IS TRUE THEN HL CONTAINS   */
    /*              THE ADDRESS OF TERM$STATUS.      */
    /*************************************************/
        A=T; CALL [GET$TERM$STATUS];
        IF (A::[IBUFF$EMPTY]) !ZERO THEN
            DO; /* INPUT BUFFER HAS NOT YET BEEN     */
                /*PROCESSED;DO NOT ACCEPT NEW BUFFER*/
            A=T; CALL [SEND$BEEP];
            A=[FALSE];
            END
        ELSE
            A=[TRUE];
        END ACCEPT$INPUT;


    CHECK$LEFT$MARGIN: PROCEDURE;
    /*************************************************/
    /* CHECKS TO SEE IF CURRENT LINE IS EMPTY.       */
    /* COMPARE$PTRS RETURNS THE APPROPRIATE TRUE/    */
    /* FALSE VALUE IN THE A REGISTER.                */
    /* INPUT: DE - ADDRESS OF CURSOR                 */
```

```
/*          BC - COMPUTED OFFSET OF TERM NBR        */
/* OUTPUT: A - RETURNED TRUE IF CURSOR IS           */
/*              PRESENTLY AT LEFT MARGIN.           */
/**************************************************/
    HL=[CURRENT$LINE]+BC;/*HL=ADD OF CURRENTLINE*/
    CALL [COMPARE$PTRS];     /* COMPARE           */
                     /* CURSOR = CURRENT$LINE */
    END CHECK$LEFT$MARGIN;


CLEAR$PTR: PROCEDURE;
/**************************************************/
/* SETS THE VALUE TO THE SPECIFIED DISPLAY        */
/* POINTER TO ZERO.                               */
/* INPUT: HL - ADDRESS OF THE DISPLAY PTR         */
/**************************************************/
    M(HL)=(A=0); HL=HL+1; M(HL)=A;
    END CLEAR$PTR;


DELETE$CHAR: PROCEDURE;
/**************************************************/
/* DECREMENTS THE CURRENT CURSOR POSITION AND     */
/* SETS NEW CURSOR POSITION DISPLAY TO BLANK.     */
/* INPUT: DE - ADDRESS OF CURSOR                  */
/*        BC - COMPUTED OFFSET OF TERMINAL NBR    */
/**************************************************/
    CALL [GET$VALUE];         /*  GET CURSOR      */
    HL=HL-1;              /*  DECREMENT CURSOR    */
    CALL [STORE$VALUE]; /* SAVE NEW CURSOR POS    */
    CALL [GET$DISPLAY$ADDR]; /* REPLACE PRESENT */
    M(HL)=(A= [BLANK]); /*      CHAR WITH BLANK */
    END DELETE$CHAR;

TERMINATE$CL: PROCEDURE;
/**************************************************/
/* TERMINATE THE CURRENT LINE.  THE SAME          */
/* PROCESSING IS DONE FOR BOTH AN MTS CMD AND     */
/* A CARRIAGE RETURN (CR) SINCE EACH SPECIFIES    */
/* THE END OF INPUT BY THE USER.                  */
/**************************************************/
            /* CHECK CHAR PRESENTLY BEING         */
            /* DISPLAYED AT CURSOR POSITION       */
            /* PRIOR TO UPDATING PTRS.            */
        A=T; CALL [CHECK$CURSOR];
            /* END OF CURRENT LINE; UPDATE        */
            /* DISPLAY POINTERS FOR NEW INPUT     */
            /* BUFFER AND NEW CURRENT LINE.       */
            /* SET END$IBUFF=CURRENT CURSOR POS */
        A=T; HL=[END$IBUFF];
        CALL [GET$INDEX];
        HL=[CURSOR]+BC;
        BC=DE; DE=HL;
        CALL [GET$VALUE];
        DE=BC+1;
        CALL [STORE$VALUE]; /* END$IBUFF=CURSOR */
            /* MOVE CURSOR TO BEGINNING OF        */
            /* NEXT  LINE. HL CONTAINS THE        */
            /* CURRENT CURSOR POSITION.           */
        IF (A=CHAR; A::[ENTER]) !ZERO THEN
            DO; /* CHAR IS EITHER NEXT LINE OR MTS CMD */
            BC=64; HL=HL+BC;/*ADD 64 TO CURRENT POS;*/
            L=(A=L  & 0C0H);/*THEN CLEAR LOWER 6 BITS*/
            A=T;
            CALL [UPDATE$CURSOR];
                /* SET CURRENT LINE = NEW CURSOR POS*/
            A=T; HL=[CURRENT$LINE];
            CALL [GET$INDEX];
            HL=[CURSOR]+BC;
            BC=DE; DE=HL;
            CALL [GET$VALUE];
            DE=BC+1;
            CALL [STORE$VALUE];/*CURRENT$LINE=CURSOR*/
```

```
            END;
    END TERMINATE$CL;


MTS$CMD: PROCEDURE;
/*************************************************/
/* CHECK TO SEE IF THIS INPUT CAN BE ACCEPTED.   */
/* IF SO, SET TERM$STATUS TO MTS$CMD$READY AND   */
/* SET MCP BIT IN TASK CONTROL TABLE TO ENSURE   */
/* MCP IS CALLED BY THE MONITOR TO PROCESS THIS */
/* MTS COMMAND.                                  */
/*************************************************/
    CALL ACCEPT$INPUT;
    IF (A=>>A) CY THEN
        DO;          /* ACCEPT INPUT BUFFER      */
        M(HL)=(A=[MTS$CMD$READY]);
        B=0; C=(A=T);
        HL=[TCT$STATUS] + BC;
        M(HL)=(A=M(HL) \ 2);
        CALL TERMINATE$CL;
        END;
    END MTS$CMD;



CARRIAGE$RETURN: PROCEDURE;
/*************************************************/
/* USER HAS TERMINATE CURRENT LINE.  CHECK TO    */
/* SEE IF NEW INPUT BUFFER CAN BE ACCEPTED.  IF */
/* SO, SET TERM$STATUS TO INPUT$WAITING AND      */
/* TERMINATE CURRENT LINE.                       */
/*************************************************/
    CALL ACCEPT$INPUT;
    IF (A=>>A) CY THEN
        DO;          /*  ACCEPT INPUT BUFFER     */
        M(HL)=(A=[INPUT$WAITING]);
        CALL TERMINATE$CL;
        END;
    END CARRIAGE$RETURN;


CHAR$DELETE: PROCEDURE;
/*************************************************/
/* CHECK TO ENSURE THAT CURRENT LINE IS NOT      */
/* EMPTY. THEN DELETE THE PREVIOUSLY ENTERED     */
/* CHAR.                                         */
/* INPUT: DE - CURSOR OFFSET ADDRESS             */
/*************************************************/
    CALL CHECK$LEFT$MARGIN;
    IF (A=>>A) CY THEN
        DO;               /* CURRENT LINE EMPTY  */
        A=T;
        CALL [SEND$BEEP];
        END
    ELSE
        DO;               /*      DELETE CHAR    */
        A=T; CALL [CHECK$CURSOR];
        A=T; HL=[CURSOR];
        CALL [GET$INDEX];
        CALL DELETE$CHAR;
        END;
    END CHAR$DELETE;


LINE$DELETE: PROCEDURE;
/*************************************************/
/* CHECK TO ENSURE THAT CURRENT LINE IS NOT      */
/* EMPTY.  IF NOT, THEN DELETE THE CURRENT LINE.*/
/* INPUT: DE - CURSOR ADDRESS OFFSET             */
/*************************************************/
    CALL CHECK$LEFT$MARGIN;
    IF (A=>>A) CY THEN
        DO;               /* CURRENT LINE EMPTY  */
```

```
        A=T;
        CALL [SEND$BEEP];
        END
    ELSE
        DO;
        A=T; CALL [CHECK$CURSOR]; A=0;
        DO WHILE (A=>>A) !CY;
        A=T; HL=[CURSOR];
        CALL [GET$INDEX]; /* GET CURSOR OFFSET  */
        CALL DELETE$CHAR;
        A=T; HL=[CURSOR];
        CALL [GET$INDEX];
        CALL CHECK$LEFT$MARGIN;
        END;
        END;
    END LINE$DELETE;


CAPITAL: PROCEDURE;
/*************************************************/
/* SET OR CLEAR THIS TERMINALS CAPITALIZE FLAG. */
/*************************************************/
    B=0; C=(A=T); HL=[CAPITALIZE]+BC;
    M(HL)= (A=M(HL) \\1);
    END CAPITAL;


CLEAR$SCREEN: PROCEDURE;
/*************************************************/
/* CLEARS THE 512 BYTE DISPLAY BUFFER AND       */
/* REINITIALIZES THE DISPLAY POINTERS.          */
/* INPUT: HL - CURSOR ADDRESS                   */
/*************************************************/
    CALL CLEAR$PTR;       /* REINITIALIZE DISPLAY */
    A=T; HL=[CURRENT$LINE];/* PTRS AND TERMINAL */
    CALL [GET$INDEX];            /* STATUS.      */
    CALL CLEAR$PTR;
    A=T; HL=[NEXT$CHAR];
    CALL [GET$INDEX];
    CALL CLEAR$PTR;
    A=T; CALL [GET$TERM$STATUS];
    M(HL)=(A=[IBUFF$EMPTY]);
                /*    CLEAR THE DISPLAY          */
    A=T; HL=[DISPLAY$BASE];
    CALL [GET$INDEX];
    CALL [GET$VALUE];  /* DISPLAY$BASE PTR IN HL*/
    BC=0; DE=512; /* SETUP INPUT PARAMETERS FOR */
                 /* [BLANK$DISPLAY] PROC        */
    CALL [BLANK$DISPLAY];
        /* RESET SWAP$POS TO CURSOR$CHAR         */
    B=0; C=(A=T);
    HL=[SWAP$POS]+BC;
    M(HL)=(A=[CURSOR$CHAR]);
    END CLEAR$SCREEN;


CURSOR$LEFT: PROCEDURE;
/*************************************************/
/* MOVES THE CURRENT CURSOR POSITION BACK ONE.  */
/* CHECKS TO ENSURE THAT CURSOR IS NOT ALREADY  */
/* AT THE LEFT MARGIN OF CURRENT LINE.          */
/* INPUT: DE - CURSOR ADDRESS                   */
/*************************************************/
    CALL CHECK$LEFT$MARGIN;
    IF (A=>>A) CY THEN
        DO;      /* AT LEFT MARGIN; SEND BEEP    */
        A=T;
        CALL [SEND$BEEP];
        END
    ELSE
        DO;      /* DECREMENT CURSOR             */
        A=T; CALL [CHECK$CURSOR];
        A=T; HL=[CURSOR];
```

```
            CALL [GET$INDEX];
            CALL [GET$VALUE];
            HL=HL-1;
            CALL [STORE$VALUE];
            END;
        END CURSOR$LEFT;


    CURSOR$RIGHT: PROCEDURE;
    /*************************************************/
    /* MOVE THE CURRENT CURSOR POSITION FORWARD ONE.*/
    /* INPUT: DE - CURSOR ADDRESS                   */
    /*************************************************/
        A=T; CALL [CHECK$CURSOR];
        A=T; HL=[CURSOR];
        CALL [GET$INDEX];
        CALL [GET$VALUE];
        HL=HL+1;            /* SETUP NEW CURSOR POS */
        A=T; CALL [UPDATE$CURSOR];
        END CURSOR$RIGHT;


    CHECK$CASE: PROCEDURE;
    /*************************************************/
    /* USES A CASE STATEMENT, INSTEAD OF 'ELSE DO'  */
    /* TO CHECK FOR THE LAST FOUR KEY COMMANDS.  IF */
    /* NOT, RESPONSE IS SET TO FALSE.  (NOTE: CASE  */
    /* STMT MUST BE USED TO GET AROUND ML80'S PARSE */
    /* STACK OVERFLOW, CAUSED BY TOO MANY 'IF THEN  */
    /* ELSE DO' STMTS.                              */
    /* INPUT: DE - CURSOR ADDRESS                   */
    /*************************************************/
        H=0; L=(A=CHAR-0A1H); /* SETUP CASE OFFSET  */
        IF (A::0) PLUS  8 (A::4) MINUS THEN
            DO CASE HL;
            CALL CAPITAL;
            CALL CURSOR$LEFT;
            CALL CURSOR$RIGHT;
            DO; HL=DE; CALL CLEAR$SCREEN; END;
            END
        ELSE
            RESPONSE=(A=[FALSE]);
        END CHECK$CASE;


    RST$CMD: PROCEDURE;
    /*************************************************/
    /* SET RST BIT IN THE TASK CONTROL TABLE SO     */
    /* RST 7 IS EXECUTED BY MONITOR.                */
    /*************************************************/
        B=0; C=(A=T);
        HL=[TCT$STATUS] + BC;
        M(HL)=(A=M(HL) \ 10H);
        END RST$CMD;

    /*************************************************/
    /* START OF KEY$COMMAND PROCESSING              */
    /*************************************************/

    T=A; CHAR=(A=C); /*  GET INPUT PARAMETERS       */
    RESPONSE=(A=[TRUE]);     /* INITIALIZE RESPONSE  */
    A=T; HL=[CURSOR];    /* GET$INDEX PARAMETERS     */
    CALL [GET$INDEX];    /* GET CURSOR OFFSET ADDRESS*/

    IF (A=CHAR-[MTS$CMD]) ZERO THEN      /* MTS CMD  */
        CALL MTS$CMD

    ELSE DO;
    IF (A=CHAR; A::[CR]) ZERO \
        (A::[ENTER]) ZERO THEN /* NEXT LINE OR     */
                               /* ENTER DEPRESSED */
        CALL CARRIAGE$RETURN
```

```
    ELSE DO;
    IF (A=CHAR-[CHAR$DELETE]) ZERO THEN
        CALL CHAR$DELETE      /*      CHAR DELETE CMD */

    ELSE DO;
    IF (A=CHAR-[LINE$DELETE]) ZERO THEN
        CALL LINE$DELETE      /*      DELETE LINE CMD */

    ELSE DO;
    IF (A=CHAR-[RST$CMD]) ZERO THEN
        CALL RST$CMD          /*      RESTART 7 CMD   */

    ELSE
        CALL CHECK$CASE;      /* USE CASE STMT TO     */
                              /* CHECK FOR REMAINING  */
                              /* KEY COMMANDS.        */

    END; END; END; END;       /* END OF ELSE DO'S     */
    A=RESPONSE;
    END KEY$COMMAND;


TERM$INPUT$CNTL: PROCEDURE;
/*****************************************************/
/* CONVERTS THE INPUT MATRIX CODE TO ASCII; CHECKS  */
/* FOR CAPIALIZATION AND CONVERTS LOWER TO UPPER    */
/* CASE LETTERS IF REQUIRED; CHECKS FOR MTS KEY     */
/* COMMANDS; IF NOT A KEY CMD THEN THE CHAR IS      */
/* DISPLAY AT THE TERMINAL AND CURSOR INCREMENTED.  */
/* INPUT: C - MATRIX CODE                           */
/*        E - TERMINAL NUMBER                       */
/* CALLED BY: TERMINAL$HLDR (INTERRUPT MOD)         */
/*****************************************************/
    DECLARE (CHAR,T) BYTE;

    T=(A=E);                 /* SAVE TERMINAL NUMBER     */

                         /* CONVERT MATRIX CODE TO ASCII */
    B=0; HL=[ASCII]+BC;
    CHAR=(A=M(HL));
                             /* CHECK FOR CAPITALIZATION */
    D=0; HL=[CAPITALIZE]+DE;
    IF (A=M(HL); A::0) !ZERO  & (A=CHAR-61H) PLUS
        & (A=CHAR-7BH) MINUS THEN /* CONVERT TO       */
            CHAR=(A=CHAR-20H); /* UPPER CASE LETTER */

                         /* CHECK FOR ANY KEY COMMANDS   */
    C=(A=CHAR); A=T;
    CALL KEY$COMMAND;

            /* A REG RETURNED TRUE IF KEY CMD FOUND */
    IF (A=>>A) !CY THEN         /* DISPLAY CHAR        */
        DO;
        A=T; CALL [CHECK$CURSOR];
        A=T; HL=[CURSOR];
        CALL [GET$INDEX]; /* GET CURSOR OFFSET         */
        CALL [GET$DISPLAY$ADDR];
        M(HL)=(A=CHAR);
            /* UPDATE CURSOR POSITION BY ONE. BC WAS*/
            /* RETURNED FROM GET$DISPLAY$ADDR SET   */
            /* TO THE VALUE OF CURSOR.              */
        HL=BC+1;
        A=T; CALL [UPDATE$CURSOR];
        END;
    END TERM$INPUT$CNTL;


EOF
/*****************************************************/
/*****  TERMINAL INTERFACE SYSTEM FUNCTIONS  ********/
/*****************************************************/
```

```
[MACRO IBUFF$EMPTY                    '0'    ]
[MACRO MTS$CMD$READY                  '0F0H' ]
[MACRO CR                             '0DH'  ]
[MACRO LF                             '0AH'  ]


/*******************************************************/
/*****  INTERNAL AND EXTERNAL LINKAGE MACROS   *******/


[INT GB TB]      [GB := 0]  [TB := 1000H]

[MACRO TASK           'M([HEX GB + 3E80H])']

[MACRO MTS$MESSAGE  '[HEX TB + 011CH]' ]
[MACRO SIZE$MESSAGE '[HEX TB + 01CFH]' ]
[MACRO CURSOR       '[HEX TB + 01D5H]' ]
[MACRO CURRENT$LINE '[HEX TB + 01DDH]' ]
[MACRO NEXT$CHAR    '[HEX TB + 01E5H]' ]
[MACRO END$IBUFF    '[HEX TB + 01EDH]' ]

[MACRO COMPARE$PTRS '[HEX TB + 0213H]' ]
[MACRO CONVERT$NUMBR$TO$ASCII '[HEX TB + 0231H]' ]
[MACRO GET$INDEX    '[HEX TB + 024CH]' ]
[MACRO GET$VALUE    '[HEX TB + 0259H]' ]
[MACRO STORE$VALUE  '[HEX TB + 0262H]' ]
[MACRO MOVE$BYTES   '[HEX TB + 026BH]' ]
[MACRO SWAP$CURSOR  '[HEX TB + 027EH]' ]

[MACRO BLANK$DISPLAY     '[HEX TB + 02A3H]']
[MACRO CHECK$CURSOR      '[HEX TB + 02D0H]']
[MACRO GET$DISPLAY$ADDR  '[HEX TB + 02B7H]']
[MACRO GET$STATUS$ADDR   '[HEX TB + 02ECH]']
[MACRO GET$TERM$STATUS   '[HEX TB + 02F9H]']
[MACRO UPDATE$CURSOR     '[HEX TB + 03C2H]']


BLINK$CURSORS: PROCEDURE;
/*******************************************************/
/* SWAPS THE CURRENT CONTENTS OF CURSOR(I) WITH       */
/* SWAP$POS(I) FOR EACH TERMINAL (I=0 TO 3).           */
/* CALLED BY: TIMER$HDLR (INTERRUPT MOD)               */
/*******************************************************/
    DECLARE I BYTE;
    I=(A=3);
    REPEAT;
        HL=[CURSOR];
        CALL [GET$INDEX];
        CALL [GET$DISPLAY$ADDR];
        A=I; CALL [SWAP$CURSOR];
        I=(A=I-1);
    UNTIL (A::0) MINUS;
    END BLINK$CURSORS;


CLEAR$STATUS$LINE: PROCEDURE;
/*******************************************************/
/* CLEARS THE STATUS LINE OF THE SPECIFIED TERMINAL.*/
/* INPUT:  A - TERMINAL NUMBER                          */
/* CALLED BY:  MTS$IPL (MONITOR MOD);                   */
/*             BUMP     (SERVICE MOD);                  */
/*******************************************************/
    CALL [GET$STATUS$ADDR];
    BC=0; DE=64;           /* SETUP PARAMETERS FOR      */
    CALL [BLANK$DISPLAY]; /* BLANK$DISPLAY PROC        */
    END CLEAR$STATUS$LINE;


MTS$MSG: PROCEDURE;
/*******************************************************/
/* CONTROLS THE MTS MESSAGE DISPLAY FIELD ON THE     */
/* STATUS LINE OF THE TERMINAL SPECIFIED BY 'TASK'. */
```

```
/* THE MTS MESSAGE FIELD STARTS AT POSITION 48 AND   */
/* UTILIZES THE REMAINING 16 BYTES FOR MTS MESSAGES */
/* (SEE MTS$MESSAGE DATA).                            */
/* INPUT:  E - MTS MESSAGE NUMBER.                    */
/* CALLED BY:  BUMP (SERVICE MOD);                    */
/*             PRINTER$HDLR (INT MOD);                */
/*             MINI$DISK (MONITOR MOD);               */
/*             RECOVER   (MONITOR MOD);               */
/*             BUMP$TASK (MONITOR MOD);               */
/****************************************************/
    DECLARE MSGNO BYTE;
    MSGNO=(A=E);
    A= [TASK];
    CALL [GET$STATUS$ADDR];
    BC=48;                  /* MTS MSG FIELD OFFSET FROM*/
                            /* STATUS BASE ADDRESS      */
    A=MSGNO; E=4; CY=0;
    REPEAT;                 /* COMPUTE OFFSET INTO THE  */
        A=<<A;              /* MTS$MESSAGE DATA VECTOR  */
    UNTIL (E=E-1) ZERO;
    DE=(HL=HL+BC);          /* SETUP PARAMETERS FOR     */
    B=0; C=A;               /* [MOVE$BYTES] PROC        */
    HL=[MTS$MESSAGE]+BC;
    BC=16; CALL [MOVE$BYTES];        /* DISPLAY MSG     */
    END MTS$MSG;


SIZE$MSG: PROCEDURE;
/****************************************************/
/* CONTROLS THE DISPLAY OF THE CURRENT MEMORY SIZE  */
/* ON THE STATUS LINE OF THE TERMINAL SPECIFIED BY  */
/* 'TASK'.  THE SIZE MESSAGE STARTS AT POSITION 40  */
/* AND HAS THE GENERAL FORMAT:                      */
/*      40     47                                   */
/*      -----------                                 */
/*       NRK MTS        E.G.  16K MTS               */
/*      -----------                                 */
/* WHERE                                            */
/*      'NR' IS THE CURRENT MEMORY SWAP SIZE        */
/*      ALLOCATED TO THAT TERMINAL USER.            */
/* THE RANGE IS FROM 0 TO 48K. THE INPUT MEMORY     */
/* SIZE NUMBER IS CONVERTED TO ASCII FOR DISPLAY.   */
/* INPUT:  A - MEMORY SIZE                          */
/* CALLED BY:  SIZE (SERVICE MOD);                  */
/*             LOGIN (SERVICE MOD);                 */
/*             RECOVER$STATUS$LINE (MONITOR MOD);   */
/****************************************************/
    DECLARE MEM$SIZE BYTE;
    MEM$SIZE=A;
    A= [TASK];
    CALL [GET$STATUS$ADDR];
    BC=40;                  /* SIZE MSG FIELD OFFSET    */
    HL=HL+BC;               /* HL=STARTING ADDRESS OF   */
                            /* SIZE MSG FIELD ON STATUS */
                            /* LINE.                    */
    A=MEM$SIZE;
    CALL [CONVERT$NUMBR$TO$ASCII];
    M(HL)=(A=B); HL=HL+1;      /* DISPLAY MEMORY SIZE   */
    M(HL)=(A=C); DE=HL+1;      /* SETUP PARAMETERS      */
    HL=[SIZE$MESSAGE];         /* FOR [MOVE$BYTES] PROC*/
    BC=6; CALL [MOVE$BYTES];/* DISPLAY REST OF        */
                               /*       SIZE MESSAGE    */
    END SIZE$MSG;


STATUS$MSG: PROCEDURE;
/****************************************************/
/* CONTROLS THE STATUS DISPLAY, POSITIONS 0 THRU 39 */
/* OF THE TERMINAL STATUS LINE.   IT HAS THE        */
/* FOLLOWING GENERAL FORMAT:                        */
/*      0                                      39   */
/*       -------------------------------------      */
/*      A=NOrB=NOrC=NOrD=NOrE=NOrF=NOrG=NOrH=NOr    */
```

```
/*      ----------------------------------------        */
/*  WHERE                                               */
/*      THE LETTER ON THE LEFT OF THE EQUAL SIGN        */
/*      SPECIFIES THE DRIVE.                            */
/*      'NO' IS THE DISK NUMBER (0-31).                 */
/*      'r' IS AN OPTIONAL PARAMETER WHICH IS           */
/*      DISPLAYED WHEN THE ATTACHED DISK IS A           */
/*      RESTRICTED (r) READ ONLY DISK.                  */
/*  THE TERMINAL IS SPECIFIED BY 'TASK'.                */
/* INPUT: A - ASCII CODE FOR RESTRICT (r),              */
/*            OR BLANK (SPACE).                         */
/*        B - DRIVE NUMBER (MUST BE CONVERTED TO        */
/*            A LETTER FOR DISPLAY).                    */
/*        C - DISK NUMBER (RANGE 0-31; MUST BE          */
/*            CONVERTED TO ASCII FOR DISPLAY).          */
/* CALLED BY: LOGIN     (SERVICE MOD);                  */
/*            ATTACH     (SERVICE MOD);                 */
/*            RECOVER$STATUS$LINE (MONITOR MOD);        */
/*******************************************************/
    DECLARE (RESTRICT,DRIVE$LTR,DISK$NR) BYTE;
                        /* GET INPUT PARAMETERS         */
    RESTRICT=A; DRIVE$LTR=(A=B); DISK$NR=(A=C);
    A= [TASK]; CALL [GET$STATUS$ADDR];
                /* COMPUTE THE APPROPRIATE STATUS       */
                /* BASE OFFSET TO DETERMINE WHERE       */
                /* TO DISPLAY THIS STATUS INFO          */
    C=0; B=(A=DRIVE$LTR);
    DO WHILE (A::0) !ZERO;
        C=(A=C+5);
        B=(A=B-1);
        END;
    HL=HL+BC;   /* SETUP ADDRESS FOR STATUS MSG.        */
                /* DISPLAY DRIVE LETTER                 */
    M(HL)=(A=DRIVE$LTR+41H);
    HL=HL+1;
                /* DISPLAY EQUAL SIGN                   */
    M(HL)=(A='=');
    HL=HL+1; A=DISK$NR;
                /* CONVERT AND DISPLAY DISK NUMBER      */
    CALL [CONVERT$NUMBR$TO$ASCII];
    M(HL)=(A=B); HL=HL+1;
    M(HL)=(A=C); HL=HL+1;
                /* DISPLAY RESTRICT OR BLANK BYTE       */
    M(HL)=(A=RESTRICT);
    END STATUS$MSG;


TERMINAL$STATUS: PROCEDURE;
/*******************************************************/
/* PROVIDES THE INTERFACE POINT FOR OTHER MTS SYSTEM*/
/* FUNCTIONS. RETRIEVES THE CURRENT TERMINAL STATUS */
/* FOR THE TERMINAL SPECIFIED BY 'TASK'.            */
/* OUTPUT: A - SET TO THE TERMINAL STATUS (EITHER   */
/*             INPUT$WAITING; MTS$CMD$READY; OR     */
/*             IBUFF$EMPTY) BY GET$TERM$STATUS PROC.*/
/* CALLED BY:  WRITE$TERMINAL; MTS(SERVICE MOD);    */
/*             READ$TERMINAL;                       */
/*             MTS$IPL (MONITOR MOD);               */
/*******************************************************/
    A=[TASK];
    CALL [GET$TERM$STATUS];
    END TERMINAL$STATUS;


READ$TERMINAL: PROCEDURE;
/*******************************************************/
/* GETS THE NEXT CHAR FROM THE TERMINAL INPUT BUFFER*/
/* SPECIFIED BY 'TASK'.                             */
/* IT IS ASSUMED THAT THE CALLING PROCEDURE HAS     */
/* CHECKED TERMINAL STATUS TO ENSURE INPUT IS       */
/* WAITING PRIOR TO CALLING READ$TERMINAL.          */
/* A TEST FOR END OF IBUFF IS MADE AND IF SO, A     */
/* 'CR' CHAR IS RETURNED; THE TERMINAL STATUS IS SET*/
```

```
/* TO EMPTY; AND THE NEXT$CHAR PTR IS SET TO CURRENT*/
/* LINE.                                             */
/* IF NOT AT END OF IBUFF, THE NEXT CHAR IS RETURNED*/
/* AND THE NEXT$CHAR PTR INCREMENTED.                */
/* OUTPUT: A - CHAR OR CR                            */
/* CALLED BY:  MTS (SERVICE MOD); MTS$IPL (MONITOR);*/
/*             MONITOR (MONITOR MOD);                */
/****************************************************/
    DECLARE CHAR BYTE;
    DECLARE PTR(2) BYTE;

    A= [TASK]; HL=[NEXT$CHAR];
    CALL [GET$INDEX];    /* DE=ADDR OF NEXT$CHAR PTR */
    HL=[END$IBUFF]+BC;/* HL=ADDR OF END$IBUFF PTR   */
    CALL [COMPARE$PTRS]; /* NEXT$CHAR=END$IBUFF ??  */
    IF (A=>>A) CY THEN
        DO;             /* AT END OF IBUFF, SET      */
                        /* NEXT$CHAR = CURRENT$LINE  */
        A= [TASK]; HL=[CURRENT$LINE];
        CALL [GET$INDEX];
        BC=(HL=[NEXT$CHAR]+BC);
        CALL [GET$VALUE];    /* HL=CURRENT$LINE VALUE*/
        DE=BC+1;
        CALL [STORE$VALUE];
                        /* UPDATE TERMINAL STATUS    */
        CALL TERMINAL$STATUS;    /* RETURNS HL=ADDR  */
                                 /* OF TERM$STATUS.  */
        M(HL)=(A= [IBUFF$EMPTY]);
                        /*   RETURN 'CR' TO CALLER   */
        CHAR=(A=[CR]);
        END
    ELSE                /* NOT AT END OF IBUFF       */
                        /* RETURN THE CHAR           */
        DO;
        A=[TASK]; HL=[NEXT$CHAR];
        CALL [GET$INDEX];    /* GET AND SAVE         */
        PTR=HL;              /* NEXT$CHAR OFFSET     */
        CALL [GET$DISPLAY$ADDR];
        CHAR=(A=M(HL));      /*  RETURN CHAR         */
                        /* INCREMENT NEXT$CHAR       */
        DE=(HL=PTR+1);       /* SETUP [STORE$VALUE]  */
        HL=BC+1;             /*   PARAMETERS         */
        CALL [STORE$VALUE];
        END;

    A=CHAR;             /* RETURN APPROPRIATE RESPONSE */
    END READ$TERMINAL;


WRITE$TERMINAL: PROCEDURE;
/****************************************************/
/* DISPLAYS THE CHAR AT THE CURRENT CURSOR POSITION */
/* OF THE TERMINAL SPECIFIED BY 'TASK'. IT CHECKS   */
/* FOR TWO SPECIAL CHARACTERS WHICH AFFECT THE      */
/* DISPLAY CURSOR POSITION.                         */
/* 'CR' RETURNS THE CURSOR TO THE BEGINNING OF THE  */
/* CURRENT DISPLAY LINE. 'LF' MOVES THE CURSOR DOWN */
/* TO THE NEXT LINE.                                */
/* FOR ALL OTHER CHARACTERS, THE CHAR IS DISPLAYED  */
/* AND THE CURSOR POSITION INCREMENTED.             */
/* PRIOR TO OUTPUT, THE CURRENT CURSOR DISPLAY      */
/* ADDRESS IS CHECKED TO ENSURE THAT THE CURSOR CHAR*/
/* IS SAVED.  OUTPUT OF CHARACTERS IS DONE UNDER    */
/* INTERRUPT LOCKOUT TO ENSURE THAT SWAPPING BY     */
/* BLINK$CURSORS PROC IS NOT DONE.                  */
/* SUBPROCEDURE: UPDATE$PTRS;                       */
/* INPUT:  E - ASCII CODE OF CHAR TO BE DISPLAYED   */
/* CALLED BY:  MTS (SERVICE MOD);                   */
/*             MTS$IPL (MONITOR MOD);               */
/****************************************************/
    DECLARE CHAR BYTE;
    DECLARE SAVE$CURSOR (2) BYTE;
```

```
UPDATE$PTRS: PROCEDURE;
/**************************************************/
/* AFTER THE DISPLAY OF EACH CHAR THE CURRENT    */
/* LINE PTR AND NEXT CHAR PTR ARE ALWAYS SET TO */
/* NEW CURSOR POSITION.  ADDITIONALLY, THE       */
/* TERMINAL'S STATUS IS SET TO IBUFF EMPTY.      */
/**************************************************/
        /*  GET CURSOR POSITION                  */
    A=[TASK]; HL=[CURSOR];
    CALL [GET$INDEX];
        /*  SET CURRENT$LINE = CURSOR            */
    BC=(HL=[CURRENT$LINE]+BC);
    CALL [GET$VALUE];    /*  HL= CURSOR VALUE    */
    DE=BC+1;
    CALL [STORE$VALUE]; /* CURRENT$LINE=CURSOR  */
    SAVE$CURSOR=HL;
               /* SET NEXT$CHAR = CURSOR        */
    A=[TASK]; HL=[NEXT$CHAR];
    CALL [GET$INDEX];
    HL=SAVE$CURSOR; DE=DE+1;
    CALL [STORE$VALUE]; /* NEXT$CHAR=CURSOR     */
               /* SET TERMINAL STATUS = EMPTY  */
    CALL TERMINAL$STATUS;
    M(HL)=(A=[IBUFF$EMPTY]);
    END UPDATE$PTRS;

    /**********************************************/
    /*  START OF WRITE$TERMINAL PROCESSING       */
    /**********************************************/

DISABLE;
CHAR=(A=E);
A=[TASK]; CALL [CHECK$CURSOR];
A=[TASK]; HL=[CURSOR];
CALL [GET$INDEX];
IF (A=CHAR-[CR]) ZERO THEN
    DO;                        /* CARRIAGE RETURN  */
    CALL [GET$VALUE];          /* HL=CURSOR        */
    L=(A=L  & 0C0H);           /* GET LEFT MARGIN      */
    END
ELSE
    DO;
    IF (A=CHAR- [LF]) ZERO THEN
        DO;                    /* LINE FEED        */
        CALL [GET$VALUE];      /* HL=CURSOR        */
        BC=64; HL=HL+BC;
        END
    ELSE                       /* DISPLAY CHAR     */
        DO;
        SAVE$CURSOR=HL;
        CALL [GET$DISPLAY$ADDR];
        M(HL)=(A=CHAR);
        DE=(HL=SAVE$CURSOR);
        CALL [GET$VALUE];
        HL=HL+1;          /* INCREMENT CURSOR      */
        END;
    END;
        /* HL REG HOLDS NEW CURSOR POSITION      */
A=[TASK]; CALL [UPDATE$CURSOR];
ENABLE;
        /* UPDATE OTHER DISPLAY PTRS              */
CALL UPDATE$PTRS;
END WRITE$TERMINAL;

EOF
```

```
/*******************************************************/
/*                    DEBUG MODULE                     */
/*******************************************************/
/*                                                     */
/* THIS MODULE PROVIDES THE FACILITY FOR INTERACTIVE   */
/* DEBUGGING OF MTS. THE DEBUGGER MODULE WAS DEVEL-    */
/* OPED TO REPLACE THE SYCOR HARDWARE DEBUGGER DUE     */
/* TO ITS QUESTIONABLE HARDWARE RELIABILITY AND LIM-   */
/* ITED DEBUG CAPABILITY.                              */
/*                                                     */
/* THE DEBUG MODULE IS DIVIDED INTO FOUR BASIC SUB-    */
/* MODULES:                                            */
/*                                                     */
/* (1) DEBUG ENTRY AND COMMAND PROCESSOR               */
/*                                                     */
/*     THIS SUBMODULE CONTAINS THE ENTRY POINT FOR     */
/*     THE DEBUGGER AND THE LOGIC TO SAVE THE MACHINE */
/*     AND MTS PROGRAM STATES. THE COMMAND PROCESSOR  */
/*     INTERPRETS THE DEBUG COMMAND REQUESTED AND      */
/*     TRANSFERS CONTROL TO THE APPROPRIATE DEBUG      */
/*     PROCESSING PROCEDURE.                           */
/*                                                     */
/* (2) DEBUG UTILITIES                                 */
/*                                                     */
/*     THIS SUBMODULE CONTAINS ALL UTILITY PROCEDURES */
/*     USED BY THE DEBUGGER.                           */
/*                                                     */
/* (3) DEBUG PROCESSING PROCEDURES                     */
/*                                                     */
/*     THIS SUBMODULE CONTAINS THE ROUTINES TO PRO-    */
/*     CESS THE FOLLOWING DEBUG COMMANDS: DISPLAY (D) */
/*     FILL (F), GO (G), GO WITH INTERUPTS DISABLED    */
/*     (H), MOVE (M), SET (S) AND EXAMINE REGISTERS    */
/*     (X). THE EXIT POINTS FROM THE DEBUGGER ARE      */
/*     CONTAINED WITHIN THE G AND H ROUTINES.          */
/*                                                     */
/* (4) DEBUG PROCESSING PROCEDURES - DISK I/O          */
/*                                                     */
/*      THIS SUBMODULE CONTAINS THE ROUTINES TO PRO-   */
/*      CESS THE FOLLOWING DEBUG COMMANDS: MINI-DISK   */
/*      BINARY INPUT (I), MINI-DISK BINARY OUTPUT (O) */
/*      AND READ HEX FORMAT (R).                       */
/*                                                     */
/*                                                     */
/* DEBUG COMMANDS                                      */
/*                                                     */
/* THE DEBUGGER CAN BE ENTERED AT ANY TIME BY PROGRAM */
/* EXECUTION OF A RST 2 INSTRUCTION. WHEN THE DEBUG-   */
/* ER IS EXECUTING THE OPERATOR IS PROMPTED WITH A ~.  */
/* THE ~ INDICATES TO THE OPERATOR THAT THE DEBUGGER   */
/* IS READY TO PROCESS A DEBUG COMMAND.                */
/* CERTAIN CONVENTIONS APPLY TO ALL DEBUG COMMANDS.    */
/* THE COMMAND DELIMITER IS EITHER A COMMA OR SPACE.   */
/* A COMMA IS SHOWN IN THE COMMAND DESCRIPTIONS WHICH */
/* FOLLOW. ANY TIME A NUMERIC PARAMETER IS ENTERED,    */
/* IT MAY BE ENTERED IN THE STANDARD HEXADECIMAL FOR-  */
/* MAT OR IN DECIMAL BY PRECEEDING THE NUMBER WITH A   */
/* PERIOD (.).                                         */
/*                                                     */
/*                                                     */
/* COMMAND DESCRIPTIONS                                */
/*                                                     */
/* 1. D - DISPLAY                                      */
/*    THE D COMMAND DISPLAYS THE CONTENTS OF MEMORY    */
/*    IN HEXADECIMAL AND ASCII FORMATS. THE FORMATS    */
/*    OF THE COMMAND ARE                               */
/*                                                     */
/*       D                                             */
/*       D<S>                                          */
/*                                                     */
/*    IF THE D COMMAND IS USED, MEMORY IS DISPLAYED    */
/*    FROM THE CURRENT DISPLAY ADDRESS (INITIALLY      */
/*    4000H), AND CONTINUES FOR 4 DISPLAY LINES.       */
```

```
/*     EACH DISPLAY LINE CONTAINS THE DISPLAY ADDRESS, */
/*     THE DATA IN HEXADECIMAL FORMAT AND THE DATA IN  */
/*     ASCII FORMAT. A PERIOD (.) IS DISPLAYED IF THE  */
/*     ASCII CHARACTER IS NON-DISPLAYABLE. THE D<S>    */
/*     COMMAND  PERFORMS IN THE SAME MANNER AS THE D   */
/*     COMMAND EXCEPT THE DISPLAY ADDRESS IS SET TO    */
/*     <S>.                                            */
/*                                                     */
/* 2.  F - FILL                                        */
/*     THE FILL COMMAND IS USED TO INITIALIZE AN AREA  */
/*     OF MEMORY TO A CONSTANT VALUE. THE FORMAT OF    */
/*     THE COMMAND IS                                  */
/*                                                     */
/*         F<S>,D>,<C>                                 */
/*                                                     */
/*     WHERE <S> IS THE STARTING ADDRESS, <F> IS THE   */
/*     FINAL ADDRESS AND <C> IS AN 8 BIT VALUE.        */
/*                                                     */
/* 3.  G - GO (PROGRAM EXECUTE)                        */
/*     THE G COMMAND TRANSFERS CONTROL FROM THE DE-    */
/*     BUGGER TO THE SPECIFIED PROGRAM COUNTER VALUE   */
/*     WITH UP TO TWO OPTIONAL BREAKPOINTS SPECIFIED.  */
/*     THE FORMS OF THE COMMAND ARE                    */
/*                                                     */
/*         G                                           */
/*         G<S>                                        */
/*         G<S>,<B>                                    */
/*         G<S>,<B>,<C>                                */
/*         G,<B>                                       */
/*         G,<B>,<C>                                   */
/*                                                     */
/*     WHERE <S> IS THE VALUE TO WHICH THE PROGRAM     */
/*     COUNTER IS SET, AND <B> AND <C> ARE OPTIONAL    */
/*     BREAKPOINT LOCATIONS. IF <S> IS NOT SPECIFIED   */
/*     THE CURRENT VALUE OF THE PROGRAM COUNTER IS     */
/*     USED. IF NO BREAKPOINTS ARE SET, THE ONLY WAY   */
/*     TO RETURN TO THE DEBUGGER IS THE EXECUTION OF   */
/*     A RST 2 INSTRUCTION.                            */
/*                                                     */
/* 4.  H - GO (PROGRAM EXECUTE WITH INTERUPTS DISABLED)*/
/*     THE H COMMAND PERFORMS THE SAME FUNCTIONS AS    */
/*     G COMMAND EXCEPT INTERUPTS ARE DISABLED WHEN    */
/*     THE MACHINE AND MTS STATES ARE RESTORED. THIS   */
/*     COMMAND IS  ESSENTIAL TO DEBUGGING THE CRITICAL */
/*     SECTIONS OF MTS CODE. THE FORMS OF THE COMMAND  */
/*     FOLLOW:                                         */
/*                                                     */
/*         H                                           */
/*         H<S>                                        */
/*         H<S>,<B>                                    */
/*         H<S>,<B>,<C>                                */
/*         H,<B>                                       */
/*         H,<B>,<C>                                   */
/*                                                     */
/*     WHERE <C>,<B> AND <C> ARE THE SAME AS THE G     */
/*     COMMAND.                                        */
/*                                                     */
/* 5.  I - MINI-DISK BINARY INPUT                      */
/*     THE I COMMAND INPUTS 512 BYTE SECTORS FROM THE  */
/*     MINI- DISK INTO  MEMORY. THE FORMAT OF THE COM- */
/*     MAND IS                                         */
/*                                                     */
/*         I<S>,<A>,<N>                                */
/*                                                     */
/*     WHERE <S> IS THE STARTING MINI-DISK SECTOR NUM- */
/*     BER, <A> IS THE ADDRESS WHERE THE DATA IS INPUT */
/*     AND <N> IS THE NUMBER OF 512 BYTE BLOCKS TO BE  */
/*     INPUT.                                          */
/*                                                     */
/* 6.  MOVE - MOVE MEMORY                              */
/*     THE M COMMAND MOVES ONE AREA OF MEMORY TO AN-   */
/*     OTHER AREA OF MEMORY. THE FORMAT OF THE COMMAND */
/*     IS                                              */
```

```
/*                                                          */
/*         M<S>,<D>,<N>                                     */
/*                                                          */
/*     WHERE <S> IS THE START ADDRESS, <D> IS THE DES-      */
/*     TINATION ADDRESS AND <N> IS THE NUMBER OF BYTES      */
/*     TO BE MOVED.                                         */
/*                                                          */
/* 7.  O - MINI-DISK BINARY OUTPUT                          */
/*     THE O COMMAND OUTPUTS MEMORY IN 512 BYTE BLOCKS      */
/*     TO THE MINI-DISK. THE FORMAT OF THE COMMAND IS       */
/*                                                          */
/*         O<S>,<A>,<N>                                     */
/*                                                          */
/*     WHERE <S> IS THE STARTING MINI-DISK SECTOR NUM-      */
/*     BER, <A> IS THE ADDRESS WHERE THE OUTPUT DATA        */
/*     IS LOCATED AND <N> IS THE NUMBER OF 512 BYTE         */
/*     BLOCKS TO BE OUTPUT.                                 */
/*                                                          */
/* 8.  R - READ HEX FORMAT                                  */
/*     THE R COMMAND READS DATA FROM THE MINI-DISK IN       */
/*     INTEL HEX FORMAT AND LOADS MEMORY. THE FORM OF       */
/*     THE COMMAND IS                                       */
/*                                                          */
/*          R<N>                                            */
/*                                                          */
/*     THE INPUT FILE MUST HAVE THE NAME .DISK<N>           */
/*     WHERE <N> IS A NUMBER FROM 0 TO 31. THE FILE         */
/*     MUST ALSO BE CREATED AND FILLED UNDER THE SYCOR      */
/*     OPERATING SYSTEM.                                    */
/*                                                          */
/* 9.  S - SET MEMORY                                       */
/*     THE S COMMAND IS USED TO EXAMINE AND OPTIONALLY      */
/*     MODIFY MEMORY ONE BYTE AT A TIME. THE FORMAT OF      */
/*     THE COMMAND IS                                       */
/*                                                          */
/*           S<A>                                           */
/*                                                          */
/*     WHERE <A> IS THE FIRST ADDRESS TO BE EXAMINED.       */
/*     THE DEBUGGER OUTPUTS THE ADDDRESS FOLLOWED BY        */
/*     THE CONTENTS OF MEMORY AT THAT ADDRESS. IF MEM-      */
/*     ORY MODIFICATION IS DESIRED, A VALUE MAY BE          */
/*     ENTERED FOLLOWED BY A CARRIAGE RETURN (<CR>).        */
/*     IF JUST A <CR> IS ENTERED THE NEXT LOCATION IS       */
/*     OPENED. A <-> CAUSES THE PREVIOUS LOCATION TO        */
/*     BE OPENED. A </> MAY BE ENTERED TO TERMINATE         */
/*     THE SET OPERATION.                                   */
/*                                                          */
/* 10. X - EXAMINE REGISTERS                                */
/*     THE X COMMAND IS USED TO EXAMINE AND OPTIONALLY      */
/*     ALTER REGISTER CONTENTS. THE FORMS OF THE COM-       */
/*     MAND ARE                                             */
/*                                                          */
/*         X                                                */
/*         X<R>                                             */
/*                                                          */
/*     WHERE <R> IS ONE OF THE FOLLOWING REGISTER           */
/*     IDENTIFIERS: C (CARRY BIT), E (EVEN PARITY BIT),     */
/*     I (INTERDIGIT CARRY BIT), Z (ZERO BIT), M (MIN-      */
/*     US - SIGN BIT), A (ACCUMULATOR), B (BC REGISTER      */
/*     PAIR), D (DE REGISTER PAIR), H (HL REGISTER          */
/*     PAIR), S (STACK POINTER) AND P (PROGRAM              */
/*     COUNTER). ALL REGISTER CONTENTS ARE DISPLAYED        */
/*     WHEN ONLY X IS ENTERED. IF X<R> IS ENTERED,          */
/*     THE SPECIFIC REGISTERS CONTENTS ARE DISPLAYED        */
/*     AND MAY OPTIONALLY BE ALTERED BY ENTERING A          */
/*     NUMERIC VALUE FOLLOWED BY A CARRIAGE RETURN.         */
/*     IF ALTERATION IS NOT DESIRED, A <CR> WILL            */
/*     CLOSE THE REGISTER.                                  */
/*                                                          */
/**********************************************************/
/**********************************************************/
```

```
/**********************************************************/
/****************** DEBUGGER CONTROL ******************/
/**********************************************************/


/*************** INTERMODULE LINKAGE MACROS ************/

[INT DB2 DB3 DB4 M2B IB GB]
[DB2:=2EC0H] [DB3:=3070H] [M2B:=0600H]
[IB:=3800H] [DB4:=3460H] [GB:=0]
[MACRO MTS '[HEX GB + 1F06H]']
[MACRO MOVBUF '[HEX M2B + 41H]']
[MACRO SVC$STACK '[HEX GB + 3C56H]']
[MACRO INT$STACK '[HEX IB + 03H]']
[MACRO GO '[HEX DB3 + 0DH]']
[MACRO HI '[HEX DB3 + 3B2H]']
[MACRO DISPLAY '[HEX DB3 + 2BAH]']
[MACRO FILL '[HEX DB3 + 1F4H]']
[MACRO MOVE '[HEX DB3 + 34EH]']
[MACRO SET '[HEX DB3 + 23CH]']
[MACRO READ '[HEX DB4 +  187H]']
[MACRO X '[HEX DB3  + 11CH]']
[MACRO CONVERT$WRITE '[HEX DB2 + 21H]']
[MACRO CRLF '[HEX DB2 + 03H]']
[MACRO SAVE '[HEX GB + 1F03H]']
[MACRO LOCK '[HEX GB + 3C54H]']
[MACRO INPUT '[HEX DB4 + 212H]']
[MACRO OUTPUT '[HEX DB4 + 21AH]']

/*************** GENERAL PURPOSE MACROS ****************/

[INT TOP USER$TOP ITOP STOP]
[TOP:=51] [USER$TOP:=34] [ITOP:=30] [STOP:=40]
[MACRO TERMINAL$STATUS 'C=8; CALL [MTS]']
[MACRO READ$TERMINAL 'C=9; CALL [MTS]']
[MACRO WRITE$TERMINAL LTR 'C=10; E=[LTR]; CALL [MTS]']
[MACRO TRUE '0FFH']
[MACRO PROMPT '7EH']
[MACRO ERROR$CHAR '3FH']

/*************** MODULE DECLARATIONS ******************/

DECLARE (DEBUG$ENTRY,DEBUG$CMD) LABEL;
DECLARE DEBUG$STACK(51) BYTE INITIAL (0);
DECLARE SAV$INT$STACK (30) BYTE INITIAL (0);
DECLARE SAV$SS$STACK (40) BYTE INITIAL (0);
DECLARE TRAP$LOC (6) BYTE INITIAL (0,0,0,0,0,0);
DECLARE I BYTE INITIAL (0);
DECLARE SAVHL DATA (0,0);
DECLARE SAVDE DATA (0,0);
DECLARE SAV$SAVE DATA (0,0,0);

ERROR: PROCEDURE;
/**********************************************************/
/* THIS ROUTINE IS EXECUTED FOR ANY DEBUG ERROR          */
/* CONDITIONS. THE STACK POINTER VALUE IS RESET          */
/* BECAUSE OF ITS UNKNOWN STATE AT THE TIME THIS         */
/* ROUTINE IS CALLED. AN ERROR PROMPT IS DISPLAYED       */
/* AT THE TERMINAL. PROGRAM CONTROL IS PASSED TO         */
/* DEBUG$CMD FOR FURTHER COMMAND PROCESSING.             */
/* CALLED BY: DEBUG$CMD,GET$PARAM,GO,X,FILL,DISPLAY,     */
/*            SET,MOVE,INIT,CHECK$RD$PNTR,READ,          */
/*            DISK$IO                                    */
/**********************************************************/
    SP=.DEBUG$STACK([USER$TOP]);
    [WRITE$TERMINAL '[ERROR$CHAR]'];
    GOTO DEBUG$CMD;
    END ERROR;

DEBUG$ENTRY:
/**********************************************************/
/* THIS ROUTINE IS THE ENTRY POINT OF THE DEBUGGER.      */
```

```
/* PROGRAM CONTROL IS PASSED TO THIS ROUTINE WHEN A    */
/* RST 2 INSTRUCTION IS EXECUTED. THE MACHINE STATE    */
/* BEFORE THE RST INSTRUCTION WAS EXECUTED IS SAVED    */
/* FOLLOWED BY BREAKPOINT PROCESSING.                  */
/*****************************************************/

    DISABLE;
    /* SAVE HL, DE REG PAIRS */
    SAVHL=HL; HL==DE; SAVDE=HL;
    DE=STACK;  /* STORE OLD PC IN DE */
    STACK=PSW;
    HL=2 + SP; /* STORE OLD SP IN HL */
    DE=DE-1; /* PC IS DECREMENTED DUE TO RST INST. */
    PSW=STACK;
    SP=.DEBUG$STACK([TOP]);
    /* START PUSHING MACHINE STATE ONTO DEBUG$STACK */
    STACK=DE; STACK=HL;
    /* RETRIEVE HL AND STACK THEN DE AND STACK */
    HL=SAVHL;
    STACK=HL;  HL=SAVDE; STACK=HL; STACK=BC; STACK=PSW;
    /* STORE FLAG BITS OF OLD PSW IN L REG */
    DE=STACK; L=E;
    STACK=DE;
    /* DETERMINE CARRY, PARITY, AC, ZERO, AND SIGN BIT */
    /* BIT AND STORE IN DEBUG$STACK(23)-DEBUGS$STACK(18) */
    B=0; C=0;
    IF (A=>L) CY THEN B=1; A=>A;
    IF (A=>A) CY THEN C=1;
    STACK=BC;
    B=0; C=0; A=>A;
    IF (A=>A) CY THEN B=1;
    A=>A;
    IF (A=>A) CY THEN C=1;
    STACK=BC; B=0;
    IF (A=>A) CY THEN B=1;
    STACK=BC;  SP=SP+1;
    /* SAVE AWAY SYSTEM STATUS */
    HL=[SAVE];  /* GET STK PTR WHICH WAS STORED IN SAVE */
                /* DURING SERVICE CALL                  */
    SAV$SAVE(0)=(A=M(HL)); HL=HL+1;
    SAV$SAVE(1)=(A=M(HL)); HL=HL+1;
    SAV$SAVE(2)=(A=M(HL));
    /* EXECUTE AS MTS CODE */
    HL=[LOCK]; M(HL)=(A=M(HL) \ 01H);
    BC=[HEX ITOP];  /* NUMBER OF BYTES TO SAVE */
    DE=[INT$STACK]; HL=.SAV$INT$STACK;
    CALL [MOVBUF];
    BC=[HEX STOP]; /* NUMBER OF BYTES TO SAVE */
    DE=[SVC$STACK]; HL=.SAV$SS$STACK;
    CALL [MOVBUF];

    I=(A=1);  /* INITIALIZE I=1 */
    DO BY I=(A=I+3) WHILE (A=I-7) !ZERO;
        HL=.TRAP$LOC; B=0; C=(A=I); HL=HL+BC;
        IF (A=M(HL); A::0) !ZERO \
            (HL=HL+1; A=M(HL); A::0) !ZERO THEN
        DO; /* PC IS NON 0 */
            HL=.TRAP$LOC; B=0; C=(A=I-1);
            /* DE POINTS TO ADDR CONTAINING OP CODE */
            HL=HL+BC; DE=HL;
            C=(A=I); HL=.TRAP$LOC; HL=HL+BC;
            /* HL POINTS TO ADDR IN WHICH TO RESTORE */
            /* OP CODE                               */
            C=M(HL); HL=HL+1;  B=M(HL);
            M(BC)=(A=M(DE));
        END;
    END;
    ENABLE;


    I=(A=1);  /* INITIAL I VALUE IS 1 */
    DO BY I=(A=I+3) WHILE (A=I-7) !ZERO;
        HL=.TRAP$LOC;
```

```
        /* OFFSET INTO TRAP$LOC LOADED IN C */
        B=0;  C=(A=I);
        HL=HL+BC;  DE=HL;
        HL=.DEBUG$STACK([HEX TOP-2]);
        IF (B=M(HL); A=M(DE); HL=HL+1; DE=DE+1; A::B) ZERO &
            (B=M(HL); A=M(DE); A::B) ZERO THEN
        DO;
            STACK=DE;
            A='*';  [WRITE$TERMINAL 'A'];
            DE=STACK;  L=1;
            CALL [CONVERT$WRITE];
        END;
    END;


    /* ZERO OUT TRAP$LOC */  A=0;
    TRAP$LOC(0)=A; TRAP$LOC(1)=A; TRAP$LOC(2)=A;
    TRAP$LOC(3)=A; TRAP$LOC(4)=A; TRAP$LOC(5)=A;

DEBUG$CMD:
/*********************************************************/
/* THIS ROUTINE IS THE DEBUG COMMAND PROCESSOR. THE      */
/* COMMAND PROCESSOR ISSUES A CARRIAGE RETURN AND        */
/* LINE FEED FOLLOWED BY THE DEBUG PROMPT CHARACTER.     */
/* WHEN A VALID COMMAND HAS BEEN ENTERED, THE APPRO-     */
/* PRIATE PROCESSING ROUTINE IS CALLED.                  */
/*********************************************************/
    CALL [CRLF];
    [WRITE$TERMINAL '[PROMPT]'];
    REPEAT;
        [TERMINAL$STATUS];
    UNTIL (A::[TRUE]) ZERO;
    [READ$TERMINAL];
    /* IF CHARACTER < 'A' OR > 'X' IGNORE AND */
    /* GET ANOTHER COMMAND                    */
    IF (C='A'; A::C) MINUS \ (C='Y'; A::C) PLUS THEN
        GOTO DEBUG$CMD;
    L=(A=A-(H='A')); H=0;
    DO CASE HL;
        CALL ERROR;          /* A */
        CALL ERROR;          /* B */
        CALL ERROR;          /* C */
        CALL [DISPLAY];      /* D DISPLAY */
        CALL ERROR;          /* E */
        CALL [FILL];         /* F FILL */
        CALL [GO];           /* G GOTO */
        CALL [HI];           /* H - GOTO W/O INTERRUPTS */
        CALL [INPUT];        /* I MINI-DISK BINARY INPUT */
        CALL ERROR;          /* J */
        CALL ERROR;          /* K */
        CALL ERROR;          /* L */
        CALL [MOVE];         /* M MOVE */
        CALL ERROR;          /* N */
        CALL [OUTPUT];       /* O MINI-DISK BINARY OUTPUT */
        CALL ERROR;          /* P */
        CALL ERROR;          /* Q */
        CALL [READ];         /* R READ HEX FORMAT */
        CALL [SET];          /* S SET MEMORY */
        CALL ERROR;          /* T */
        CALL ERROR;          /* U */
        CALL ERROR;          /* V */
        CALL ERROR;          /* W */
        CALL [X];            /* X EXAMINE REGISTERS */
    END;                    /* CASE */
    GOTO DEBUG$CMD;

EOF


/*******************************************************/
/***************** DEBUG UTILITIES ********************/
/*******************************************************/
```

```
/************ INTERMODULE LINKAGE MACROS *************/

[INT DB1] [DB1:=2BA0H]
[MACRO MTS '1F06H']
[MACRO ERROR '[HEX DB1 + 13H]']

/************ GENERAL PURPOSE MACROS *****************/

[MACRO TERMINAL$STATUS 'C=8; CALL [MTS]']
[MACRO READ$TERMINAL 'C=9; CALL [MTS]']
[MACRO WRITE$TERMINAL LTR 'C=10; E=[LTR]; CALL [MTS]']
[MACRO TRUE '0FFH']
[MACRO FALSE '00H']
[MACRO CR '0DH']
[MACRO LF '0AH']
[MACRO SLASH '2FH']
[MACRO COMMA '2CH']
[MACRO SPACE '20H']
[MACRO PERIOD '2EH']
[MACRO COLON '3AH']
[MACRO DASH '2DH']
[MACRO WAIT$READ 'REPEAT;
                          [TERMINAL$STATUS];
                      UNTIL (A::[TRUE]) ZERO;
              [READ$TERMINAL]']

/************ MODULE DECLARATIONS *******************/

DECLARE I BYTE INITIAL (0);

CRLF: PROCEDURE;
/*****************************************************/
/* THIS ROUTINE OUTPUTS A CARRIAGE RETURN AND LINE   */
/* FEED TO THE TERMINAL.                             */
/* CALLED BY: DEBUG$CMD,DISPLAY                      */
/*****************************************************/
    [WRITE$TERMINAL '[CR]'];
    [WRITE$TERMINAL '[LF]'];
    END CRLF;

BLANK: PROCEDURE;
/*****************************************************/
/* THIS ROUTINE DISPLAYS A SPACE AT THE TERMINAL.    */
/* CALLED BY: X,SET,DISPLAY                          */
/*****************************************************/
    A=[SPACE];
    [WRITE$TERMINAL 'A'];
    END BLANK;

CONVERT$WRITE: PROCEDURE;
/******************************************************/
/* CONVERT AND WRITE BYTES TO THE TERMINAL.           */
/* INPUT: DE - ADDRESS OF FIRST BYTE TO BE WRITTEN    */
/*         L - WRITE FLAG                             */
/*             0 - ONLY WRITE 1 BYTE, DE=DE+1         */
/*             1 - WRITE THIS BYTE AND PREVIOUS,      */
/*                 DE=DE-2                            */
/*             2 - WRITE THIS BYTE AND NEXT, DE=DE+2  */
/*             3 - ONLY WRITE LOWER NIBBLE, DE=DE+1   */
/* OUTPUT: DE - MODIFIED AS DESCRIBED ABOVE           */
/* CALLED BY: DEBUG$ENTRY,X,SET,DISPLAY               */
/******************************************************/
    H=0; BC=HL;
    DO CASE HL;    /* SET CHARACTER COUNT */
        I=(A=2);   /* DATA TYPE 0 */
        I=(A=4);   /*           1 */
        I=(A=4);   /*           2 */
        I=(A=1);   /*           3 */
    END; HL=BC;
    A=M(DE);
    C=A;
    IF (A=L; A::3) !ZERO THEN
```

```
        DO; /* IF NOT DATA TYPE 3 - USE MSN */
            A=C;
            A=>>A; A=>>A; A=>>A; A=>>A;
            C=A;
        END;
    REPEAT;
        A=C & 0FH;
         /* CONVERT THE CONTENTS OF A        */
         /* (ASSUMED IN THE RANGE 0 - 15)    */
         /* TO A PRINTABLE HEX CHARACTER     */
        IF (A::10) MINUS THEN
            A=A + 30H  /* A=A-'0' */
        ELSE
            A=A - 10, + 41H;  /* A=A-10,+'A' */
        STACK=HL; STACK=DE;
        [WRITE$TERMINAL 'A'];  /* WRITE CHARACTER */
        DE=STACK; HL=STACK;
        IF (A=L; A::1) ZERO THEN
            DO;  /* DATA TYPE 1 - DECREMENT ADDR IF  */
                 /* MSN & LSN OUTPUT                 */
                I=(A=I-1); A=>>A;
                IF !CY THEN
                    DE=DE-1;
            END
        ELSE
            DO;  /* DATA TYPES 0,2 OR 3 - INCR ADDR IF */
                 /* MSN & LSN OUTPUT                   */
                I=(A=I-1); A=>>A;
                IF !CY THEN
                    DE=DE+1;
            END;
        A=M(DE);
        IF !CY THEN
            DO; /* IF MSB TO BE WRITTEN, MOVE TO LSB */
            A=>>A; A=>>A; A=>>A; A=>>A;
            END;
        C=A;
    UNTIL (A=I; A::0) ZERO;
    END CONVERT$WRITE;

GET$PARAM: PROCEDURE;
/*******************************************************/
/* THIS ROUTINE READS AND RETURNS A HEX OR DECIMAL    */
/* PARAMETER IN REGISTER HL. A PARAMETER HAS BEEN     */
/* ENTERED IF REGISTER A IS ZERO, OTHERWISE REGISTER */
/* A IS NONZERO AND CONTAINS THE NEXT TO LAST CHAR-   */
/* ACTER ENTERED. IF THE LAST CHARACTER ENTERED IS A */
/* CR THEN CY=1, OTHERWISE CY=0.                      */
/* INPUT: NONE                                        */
/* OUTPUT: CY - INDICATES WHETHER LAST CHAR WAS A CR */
/*              0 - NOT A CR                          */
/*              1 - CR ENTERED                        */
/*          A - INDICATES VALID PARAMATER ENTRY       */
/*              0 - VALID PARAM ENTERED               */
/*              NON 0 - NEXT TO LAST CHAR ENTERED     */
/*         HL - VALUE OF VALID PARAMETER              */
/* CALLED BY: GO,X,FILL,SET,DISPLAY,MOVE,READ,        */
/*            DISK$IO                                 */
/*******************************************************/
    [MACRO NOT$DELIMETER '(A::[COMMA]) !ZERO &
     (A::[SPACE]) !ZERO & (A::[CR]) !ZERO ']
    DECLARE (VALID$PARAM,HEX$FLAG) BYTE INITIAL (0,0);
    VALID$PARAM=(A=0);  /* INITIALIZE FLAGS */
    HEX$FLAG=(A=[TRUE]);
    [WAIT$READ];
    VALID$PARAM=A;
    IF (A::[SLASH]) ZERO \ (A::[DASH]) ZERO THEN
    DO; /* STRIP OFF FIRST POUND SIGN OR MINUS SIGN */
       [WAIT$READ];
    END;
    IF (A::[PERIOD]) ZERO THEN
       DO;  /* SET-UP FOR DECIMAL PARAMETER */
          VALID$PARAM=A;
```

```
            HEX$FLAG=(A=[FALSE]);
            [WAIT$READ];
        END;
    HL=0;
    DO WHILE [NOT$DELIMETER];
        /* IF (CHAR > 9 AND CHAR < 'A') OR CHAR < '0' */
        /* OR CHAR > 'F' - CALL ERROR                 */
        IF (IF (A::[COLON]) PLUS 8 (A::41H) MINUS THEN
                CY=1 ELSE CY=0)
            CY \ (A::30H) MINUS \ (A::47H) PLUS THEN
                CALL [ERROR];
        /* CONVERT ASCII CHAR TO HEX DIGIT */
        /* IF DECIMAL PROCESSING AND CHAR > 9 - CALL ERROR */
        A=A-30H;  /* A=A-'0' */
        IF (A::10) PLUS THEN
            DO;
                IF (C=A; A=HEX$FLAG; A::[FALSE]; A=C) ZERO THEN
                    CALL [ERROR];
                A=A-7;
            END;
        IF (C=A; A=HEX$FLAG; A::[TRUE]; A=C) ZERO THEN
            DO;  /* HEX CONVERSION */
                HL=HL+HL,+HL,+HL,+HL;  /* HL=HL*16 */
                A=A\L; L=A;
            END
        ELSE
            DO;  /* DECIMAL CONVERSION */
                DE=(HL=HL+HL);  /* DE=HL*2 */
                HL=HL+HL,+HL,+DE;  /* HL=HL*8 + HL*2 */
                D=0; E=A;
                HL=HL+DE;
            END;
        STACK=HL;
        VALID$PARAM=(A=0);
        [WAIT$READ];
        HL=STACK;
    END;  /* DO WHILE */
    IF (A::[CR]) ZERO THEN
        CY=1
    ELSE
        CY=0;
    A=VALID$PARAM;
    END GET$PARAM;

EOF


/******************************************************/
/************* DEBUG PROCESSING PROCEDURES *************/
/******************************************************/


/************* INTERMODULE LINKAGE MACROS *************/

[INT DB1 DB2 M2B IB GB] [DB1:=2BA0H] [DB2:=2EC0H]
                [M2B:=0600H] [IB:=3800H] [GB:=0000H]
[MACRO MTS '[HEX GB + 1F06H]']
[MACRO MOVBUF '[HEX M2B + 41H]']
[MACRO INT$STACK '[HEX IB + 03H]']
[MACRO SVC$STACK '[HEX GB + 3C56H]']
[MACRO SAV$SS$STACK '[HEX DB1 + 29AH]']
[MACRO SAV$INT$STACK '[HEX DB1 + 27CH]']
[MACRO GET$PARAM '[HEX DB2 + 0BBH]']
[MACRO ERROR '[HEX DB1 + 13H]']
[MACRO DEBUG$STACK  '[HEX DB1 + 249H]']
[INT DEBUG$STK] [DEBUG$STK:=2DE9H]
[MACRO TRAP$LOC '[HEX DB1 + 2C2H]']
[MACRO BLANK '[HEX DB2 + 15H]']
[MACRO CONVERT$WRITE '[HEX DB2 + 21H]']
[MACRO CRLF '[HEX DB2 + 03H]']
[MACRO SAVE '[HEX GB + 1F03H]']
[MACRO SAV$SAVE '[HEX DB1 + 0DH]']
[MACRO LOCK '[HEX GB + 3C54H]']
```

```
/************** GENERAL PURPOSE MACROS *****************/

[INT TOP USER$TOP ITOP STOP]
[TOP:=51] [USER$TOP:=34] [ITOP:=30] [STOP:=40]
[MACRO READ$TERMINAL 'C=9; CALL [MTS]']
[MACRO WRITE$TERMINAL LTR 'C=10; E=[LTR]; CALL [MTS]']
[MACRO TRUE '0FFH']
[MACRO FALSE '00H']
[MACRO CR '0DH']

/************** MODULE DECLARATIONS *******************/

DECLARE (SAVCY,I,J,EXIT,HFLAG) BYTE INITIAL (0,0,0,0,0);
DECLARE SAVHL DATA (0,0);
DECLARE SAVDE DATA (0,0);

GO: PROCEDURE;
/********************************************************/
/* THIS ROUTINE PROCESSES THE G COMMAND AND IS THE     */
/* EXIT POINT FROM THE DEBUGGER.                        */
/* CALLED BY: DEBUG$CMD,HI                              */
/********************************************************/


    SAVCY=(A=0);  I=(A=0);
    REPEAT;
        CALL [GET$PARAM];
        D=A; /* SAVE ACCUMULATER */
        IF CY THEN SAVCY=(A=1);
        IF (A=I; A::0) ZERO THEN
        DO; /* RETURN WITH FIRST PARAM */
            IF (A=D; A::0) ZERO THEN
            DO; /* STORE START POINT */
                DE=HL; H=0; L=[HEX TOP-1];
                BC=[DEBUG$STACK]; HL=HL+BC;
                M(HL)=(A=D); HL=HL-1;
                M(HL)=(A=E);
            END;
        END
        ELSE
        IF (A=I; A::1) ZERO \ (A=I; A::4) ZERO THEN
        /* BREAK POINTS HAVE BEEN SPECIFIED */
        DO;
            HL==DE; /* DE NOW CONTAINS BREAK PT. TO STORE */
            HL=[TRAP$LOC];
            B=0; C=(A=I);
            /* STORE LO BYTE OF BK PT */
            HL=HL+BC; M(HL)=(A=E);
            /* STORE HI BYTE OF BK PT */
            HL=HL+1; M(HL)=(A=D);
            /* STORE OP CODE */
            HL=HL-1,-1; M(HL)=(A=M(DE));
            M(DE)=(A=0D7H);
            I=(A=I+2); /* ADJUST INDEX */
        END
        ELSE
        IF (A=D; A::0) !ZERO THEN CALL [ERROR];

        I=(A=I+1); /* ADJUST INDEX */
    UNTIL (A=SAVCY; A::1) ZERO;

/********************************************************/
/* THE FLAG BYTE OF THE PSW IS UPDATED WITH CHANGES    */
/* WHICH MAY HAVE BEEN MADE WHILE IN THE DEBUGGER      */
/* FLAG VALUES ARE STORED INTO THE BC AND DE REGIS-    */
/* TER PAIRS.  THE PSW FLAG BYTE IS STORED IN THE      */
/* H REGISTER PRIOR TO UPDATING ITS VALUES             */
/********************************************************/

    DISABLE;
    /* RESTORE SYSTEM STATUS */
        BC=[HEX ITOP]; /* # OF BYTES TO MOVE */
```

```
        DE=[SAV$INT$STACK]; /* START ADDR */
        HL=[INT$STACK]; /* DEST ADDR */
        CALL [MOVBUF];
        BC=[HEX STOP];
        DE=[SAV$SS$STACK];
        HL=[SVC$STACK];
        CALL [MOVBUF];
        HL=[SAVE];  DE=[SAV$SAVE];
        M(HL)=(A=M(DE));  HL=HL+1;  DE=DE+1;
        M(HL)=(A=M(DE));  HL=HL+1;  DE=DE+1;
        /* STK PTR SAVED IN SVC CALL RESTORED */
        M(HL)=(A=M(DE));
        /* RESET LOCK BYTE */
        HL=[LOCK]; M(HL)=(A=M(HL) & 0FEH);
    /* ADJUST STACK POINTER */
    SP=[HEX DEBUG$STK + USER$TOP];
    BC=STACK;  DE=STACK;  HL=STACK;
    A=>>C;  A=<H;  H=A;   /* LOAD SIGN BIT */
    A=>>B;  A=<H;  /* LOAD ZERO BIT */
    A=<<A;  H=A;  /* ADJUST UNUSED BIT */
    A=>>E;  A=<H;  /* LOAD AC BIT */
    A=<<A;  H=A;  /* ADJUST UNUSED BIT */
    A=>>D;  A=<H;   /* LOAD PARITY BIT */
    A=<<A;  H=A;  /* ADJUST UNUSED BIT */
    A=>>L;  A=<H; B=A;  /* LOAD CARRY BIT, STORE UPDATED */
                        /* FLAG BYTE OF PSW IN E. */
    STACK=BC;  SP=SP+1;

/* POP  MACHINE STATE AND OLD SP AND PC      */

    PSW=STACK;
    BC=STACK;  DE=STACK;  HL=DE;
    SAVDE=HL;  HL=STACK; SAVHL=HL;
    HL=STACK;  DE=STACK;  /* GET SP AND PC */
    SP=HL;  STACK=DE;
    HL=SAVDE;  DE=HL;  HL=SAVHL;
    STACK=PSW;
    IF (A=HFLAG; A::[FALSE]) ZERO THEN /* HFLAG NOT SET */
        ENABLE;
    HFLAG=(A=[FALSE]);
    PSW=STACK; RETURN; /*POP PC OFF STACK */
    END GO;

SET$UP: PROCEDURE;
/**********************************************************/
/* ROUTINE TO LOAD REGISTER DE WITH ADDRESS OF BYTE      */
/* OR BYTES TO BE DISPLAYED AND REGISTER L TO TYPE OF    */
/* DISPLAY.                                              */
/* CALLED BY: X                                          */
/**********************************************************/
    BC=BC+1;  E=(A=M(BC));
    BC=BC+1;  D=(A=M(BC));
    BC=BC+1;  L=(A=M(BC));
    BC=BC+1;  /* POSITION POINTER TO NEXT ENTRY */
    END SET$UP;

X: PROCEDURE;
/**********************************************************/
/* THIS ROUTINE IS USED TO EXAMINE AND OPTIONALLY        */
/* MODIFY REGISTERS.                                     */
/* CALLED BY: DEBUG$CMD                                  */
/**********************************************************/
    /* REGISTER POINTER TABLE - POINTS TO LOCATION IN */
    /* DEBUG$STACK FOR REGISTER CONTENTS. FIRST ENTRY */
    /* IS ASCII CODE CODE FOR REG ID, SECOND ENTRY IS */
    /* DEBUG$STACK ADDRESS AND THIRD ENTRY IS REG     */
    /* TYPE - 3 FOR NIBBLE, 0 FOR BYTE AND 1 FOR TWO  */
    /* BYTES.                                         */
    DECLARE REG$PNTR(44) BYTE INITIAL
        /* C */(43H,[HEX DEBUG$STK+TOP-13],3,
        /* E */ 45H,[HEX DEBUG$STK+TOP-14],3,
        /* I */ 49H,[HEX DEBUG$STK+TOP-15],3,
        /* Z */ 5AH,[HEX DEBUG$STK+TOP-16],3,
```

```
      /* M */ 4DH,[HEX DEBUG$STK+TOP-17],3,
      /* A */ 41H,[HEX DEBUG$STK+TOP-11],0,
      /* B */ 42H,[HEX DEBUG$STK+TOP-9],1,
      /* D */ 44H,[HEX DEBUG$STK+TOP-7],1,
      /* H */ 48H,[HEX DEBUG$STK+TOP-5],1,
      /* S */ 53H,[HEX DEBUG$STK+TOP-3],1,
      /* P */ 50H,[HEX DEBUG$STK+TOP-1],1);

[READ$TERMINAL];  /* CHECK FOR CR */
IF (A::[CR]) ZERO THEN
   DO;  /* CR ENTERED - DISPLAY ALL REGISTERS */
      BC=.REG$PNTR(0);
      STACK=BC;
      I=(A=5);
      REPEAT;  /* DISPLAY FLAG BITS */
         BC=STACK;
         A=M(BC);
         STACK=BC;
         [WRITE$TERMINAL 'A'];
         BC=STACK;
         CALL SET$UP;
         STACK=BC;
         CALL [CONVERT$WRITE];
      UNTIL (I=(A=I-1); A::0) ZERO;
      I=(A=6);
      REPEAT;  /* DISPLAY REGISTER VALUES */
         CALL [BLANK];
         BC=STACK;
         A=M(BC);
         STACK=BC;
         [WRITE$TERMINAL 'A'];
         A='='; [WRITE$TERMINAL 'A'];
         BC=STACK;
         CALL SET$UP;
         STACK=BC;
         CALL [CONVERT$WRITE];
      UNTIL (I=(A=I-1); A::0) ZERO;
      BC=STACK;
   END  /* DISPLAY ALL REGISTERS */
ELSE
   DO;  /* REGISTER MODIFICATION */
      E=A;
      EXIT=(A=[FALSE]);
      DO BC=.REG$PNTR(0) BY BC=BC+1,+1,+1,+1
        WHILE (A=EXIT; A::[FALSE]) ZERO &
              (HL=.REG$PNTR(43); A=L-C; A=H--B) PLUS;
         IF (A=M(BC); A::E) ZERO THEN
            DO;  /* REGISTER MATCH */
               STACK=BC;
               [WRITE$TERMINAL 'A'];
               A='='; [WRITE$TERMINAL 'A'];
               BC=STACK; STACK=BC;
               CALL SET$UP;
               CALL [CONVERT$WRITE];
               CALL [BLANK];
               CALL [GET$PARAM];
               IF !CY THEN
                  CALL [ERROR];  /* NO CR ENTERED */
               BC=STACK;
               IF (A::0) ZERO THEN
                  DO;  /* STORE REG VALUE */
                     STACK=HL;
                     CALL SET$UP;
                     BC=STACK;  /* RESTORE NEW VALUE */
                     IF (A=L; A::1) ZERO THEN
                        DO;
                           M(DE)=(A=B);
                           DE=DE-1;
                        END;
                     M(DE)=(A=C);
                  END;
               EXIT=(A=[TRUE]);
            END;  /* REGISTER MATCH */
```

```
        END; /* DO WHILE */
        IF (A=EXIT; A::[FALSE]) ZERO THEN
           CALL [ERROR];  /* NO REGISTER MATCH FOUND */
      END; /* REGISTER MODIFICATION */
   END X;

FILL: PROCEDURE;
/*****************************************************/
/* THIS ROUTINE IS USED TO SET A MEMORY BLOCK TO A   */
/* SPECIFIED VALUE.  THE ROUTINE INPUTS THE THREE    */
/* REQUIRED PARAMETERS: START ADDRESS, FINAL ADD-    */
/* RESS, AND BYTE CONSTANT.  THE CONSTANT IS THEN    */
/* PLACED IN THE START ADDRESS THRU THE FINAL ADD-   */
/* RESS.                                             */
/* CALLED BY: DEBUG$CMD                              */
/*****************************************************/
   CALL [GET$PARAM];  /* INPUT START ADDRESS */
   IF CY \ (A::0) !ZERO THEN
      CALL [ERROR];
   STACK=HL;
   CALL [GET$PARAM]; /* INPUT FINAL ADDRESS */
   IF CY \ (A::0) !ZERO THEN
      CALL [ERROR];
   STACK=HL;
   CALL [GET$PARAM]; /* INPUT BYTE CONSTANT */
   IF !CY \ (A::0) !ZERO THEN
      CALL [ERROR];
   D=L; HL=STACK; BC=STACK;
   /* HL=FINAL ADDR, BC=START ADDR */
   IF (A=L-C; A=H--B) MINUS THEN
      CALL [ERROR]; /* FINAL ADDR < START ADDR */
   REPEAT;  /* PERFORM FILL */
      A=D;
      M(BC)=A;
      BC=BC+1; D=A;
   /* FINAL ADDR < START ADDR */
   UNTIL (A=L-C; A=H--B) MINUS;
   END FILL;

SET: PROCEDURE;
/*******************************************************/
/* ROUTINE WHICH ALLOWS MEMORY LOCATIONS TO BE SET     */
/* AND OPTIONALLY ALTERED. THE ROUTINE INPUTS ONE      */
/* REQUIRED PARAMETER WHICH IS THE ADDRESS AT WHICH    */
/* MEMORY EXAMINATION IS TO BEGIN.                     */
/* CALLED BY: DEBUG$CMD                                */
/*******************************************************/
   DECLARE SET$LOC(2) BYTE INITIAL (0,0);
   EXIT=(A=[FALSE]);
   CALL [GET$PARAM];  /* GET SET LOCATION */
   IF !CY \ (A::0) !ZERO THEN
      CALL [ERROR]; /* ERROR IF NO SET LOCATION ENTERED */
   SET$LOC=HL;  /* SAVE SET LOCATION */
   DO WHILE (A=EXIT; A::[FALSE]) ZERO;
      /* WRITE OUT ADDRESS FOLLOWED BY CONTENTS */
      DE=.SET$LOC(1); L=1; CALL [CONVERT$WRITE];
      CALL [BLANK];
      HL=SET$LOC; DE=HL; L=0; CALL [CONVERT$WRITE];
      CALL [BLANK];
      /* PROCESS INPUT */
      CALL [GET$PARAM];
      IF !CY THEN  /* NO CR ENTERED - ERROR */
         CALL [ERROR];
      IF (A::0) ZERO THEN
         DO;  /* VALUE ENTERED - SAVE IT */
            DE=HL;
            HL=SET$LOC;
            M(HL)=E; SET$LOC=(HL=SET$LOC+1);
         END
      ELSE
         DO;  /* SLASH, CR OR MINUS ENTERED */
            IF (A::2FH) ZERO THEN  /* SLASH ENTERED - EXIT */
               EXIT=(A=[TRUE])
```

```
            ELSE
               DO;  /* DECR SET LOC IF MINUS */
                    /* ELSE INCR SET LOC     */
                  IF (A::2DH) ZERO THEN
                     SET$LOC=(HL=SET$LOC-1)
                  ELSE
                     SET$LOC=(HL=SET$LOC+1);
               END;
         END;
   END;  /* DO WHILE */
   END SET;

DISPLAY: PROCEDURE;
/*********************************************************/
/* THIS ROUTINE DISPLAYS 4 LINES OF MEMORY IN HEXA-     */
/* DECIMAL AND ASCII FORMATS. EACH LINE DISPLAYS 16     */
/* BYTES OF DATA. MEMORY IS DISPLAYED FROM THE          */
/* CURRENT DISPLAY LINE. THE DISPLAY LINE IS AUTO-      */
/* MATICALLY UPDATED BY THE PROCEDURE AS MEMORY IS      */
/* DISPLAYED OR MAY BE SET BY THE COMMAND DS WHERE      */
/* S IS THE NEW DISPLAY LINE.                           */
/* CALLED BY: DEBUG$CMD                                 */
/*********************************************************/
   DECLARE DISPLAY$LINE(2) BYTE INITIAL (00H,40H);
   CALL [GET$PARAM];
   IF !CY THEN  /* CR WAS NOT ENTERED AS DELIMETER */
      CALL [ERROR];
   IF (A::0) ZERO THEN  /* SET NEW DISPLAY LINE */
      DISPLAY$LINE=HL;
   I=(A=4);  /* SET LINE COUNTER */
   REPEAT;   /* DISPLAY LINE */
      HL=.DISPLAY$LINE(1);
      DE=HL; L=1;
      CALL [CONVERT$WRITE];  /* WRITE ADDRESS */
      CALL [BLANK]; CALL [BLANK];
      HL=DISPLAY$LINE;
      DE=HL; L=2;
      J=(A=8);
      REPEAT;  /* WRITE OUT HEX DATA */
         CALL [CONVERT$WRITE];
         STACK=HL; STACK=DE;
         CALL [BLANK];
         DE=STACK; HL=STACK;
      UNTIL (J=(A=J-1); A::0) ZERO;
      CALL [BLANK];
      HL=DISPLAY$LINE;
      J=(A=16);
      REPEAT;  /* WRITE OUT ASCII DATA */
         A=M(HL),&7FH;
         /* PRINT A PERIOD FOR CHARS > SPACE OR RUBOUT */
         IF (A::20H) MINUS \ (A::7FH) ZERO THEN
            A='.';
         STACK=HL;
         [WRITE$TERMINAL 'A'];
         HL=STACK,+1;
      UNTIL (J=(A=J-1); A::0) ZERO;
      CALL [CRLF];
      HL=DISPLAY$LINE,+(DE=16);
      DISPLAY$LINE=HL;
   UNTIL (I=(A=I-1); A::0) ZERO;
   END DISPLAY;


MOVE: PROCEDURE;
/*********************************************************/
/* THIS ROUTINE MOVES A SPECIFIED NUMBER OF BYTES FROM*/
/* ONE SECTION OF MEMORY SPECIFIED BY THE USER TO AN- */
/* OTHER SECTION.  GET$PARAM IS CALLED TO DETERMINE   */
/* THE START AND DESTINATION ADDRESSES AS WELL AS THE */
/* NUMBER OF BYTES TO BE MOVED.                       */
/* CALLED BY: DEBUG$CMD.                              */
/*********************************************************/
```

```
    I=(A=0); SAVCY=A;
    REPEAT;
         CALL [GET$PARAM];
         IF CY THEN SAVCY=(A=1);
         IF (A=I; A::0) ZERO THEN
         STACK=HL /* SAVE START ADDRESS */
         ELSE
         IF (A=I; A::1) ZERO THEN
         STACK=HL /* SAVE DESTINATION ADDRESS */
         ELSE
         IF (A=I; A::2) ZERO THEN
         DO;
              B=H; C=L; /* BC GETS # OF BYTES TO BE MOVED */
              HL=STACK; /* HL GETS DESTINATION ADDRESS */
              DE=STACK; /* DE GETS START ADDRESS */
              REPEAT;
                   M(HL)=(A=M(DE));
                   HL=HL+1;  DE=DE+1;
              UNTIL (BC=BC-1; A=0; A::C) ZERO & (A::B) ZERO;
         END;
         I=(A=I+1);
    UNTIL (A=SAVCY; A::1) ZERO;
    IF (A=I; A::3) !ZERO THEN CALL [ERROR];
    END MOVE;

HI: PROCEDURE;
/*************************************************/
/* THE H ROUTINE SETS A FLAG CALLED HFLAG WHICH  */
/* CAUSES AN EXIT FROM THE GO ROUTINE WITH IN-   */
/* TERRUPTS DISABLED.  THIS PROCEDURE IS NECES-  */
/* SARY FOR THE DEBUGGING OF CRITICAL SECTIONS   */
/* OF CODE; THAT IS PORTIONS IN WHICH THE INTER- */
/* RUPT HANDLER HAS BEEN DISABLED.               */
/* CALLED BY: DEBUG$CMD                          */
/*************************************************/
    HFLAG=(A=[TRUE]);
    CALL GO;
    END HI;

EOF


/*****************************************************/
/******* DEBUG PROCESSING PROCEDURES - DISK I/O *******/
/*****************************************************/


/*************** INTERMODULE LINKAGE MACROS ***********/

[INT DB1 DB2 M2B GB]
[M2B:=0600H] [DB1:=2BA0H] [DB2:=2EC0H] [GB:=0]
[MACRO MTS '1F06H']
[MACRO ERROR '[HEX DB1 + 13H]']
[MACRO GET$PARAM '[HEX DB2 + 0BBH]']
[MACRO DMT$FLAG '[HEX GB + 3EBDH]']
[MACRO DMT$BOE '[HEX GB + 3EDDH]']
[MACRO DMT$EOE '[HEX GB + 3F1DH]']
[MACRO MDBUF '[HEX GB + 3C7EH]']
[MACRO MDBUF$MAX '[HEX GB + 3C7EH + 512]']
[MACRO MINI$DISK '[HEX M2B + 54H]']

/*************** GENERAL PURPOSE MACROS ***************/

[MACRO WRITE$TERMINAL LTR 'C=10; E=[LTR]; CALL [MTS]']
[MACRO SUB$HL$DE 'L=(A=L-E); H=(A=H--D)']
[MACRO TRUE '0FFH']
[MACRO FALSE '00H']

/**************** MODULE DECLARATIONS *****************/

DECLARE CHAR$PNTR(2) BYTE INITIAL (0,0);
DECLARE REC$PNTR(2) BYTE INITIAL (0,0);
DECLARE C$SECTOR(2) BYTE INITIAL (0,0);
```

```
DECLARE RECSLEN BYTE INITIAL (0);
DECLARE BOE(2) BYTE INITIAL (0,0);
DECLARE EOE(2) BYTE INITIAL (0,0);
DECLARE EXIT BYTE INITIAL (0);

INIT: PROCEDURE;
/*********************************************************/
/* ROUTINE WHICH VERIFIES THAT DISK EXISTS AND SETS     */
/* BOE AND EOE OF DISK INTO BOE AND EOE VARIABLES.      */
/* INPUT: L - DISK NUMBER                               */
/* OUTPUT: BOE AND EOE VARIABLES SET                    */
/* CALLED BY: READ                                      */
/*********************************************************/
    IF (A=L; A::32) PLUS THEN  /* INVALID DISK NUMBER */
       CALL [ERROR];
    H=0; C=L;
    DE=[DMT$FLAG]; HL=HL+DE;
    IF (A=>M(HL)) !CY THEN  /* DISK DOESN'T EXIST */
       CALL [ERROR];
    /* COMPUTE BOE AND EOE */
    H=0; L=(C=(A=<<C));
    DE=[DMT$BOE]; HL=HL+DE;
    BOE(0)=(A=M(HL)); HL=HL+1; BOE(1)=(A=M(HL));
    H=0; L=C;
    DE=[DMT$EOE]; HL=HL+DE;
    EOE(0)=(A=M(HL)); HL=HL+1; EOE(1)=(A=M(HL));
    END INIT;

CHECK$RD$PNTR: PROCEDURE;
/*********************************************************/
/* ROUTINE TO CHECK THE CHARACTER POINTER. IF CHAR$     */
/* PNTR EXCEEDS THE BUFFER LENGTH, THE NEXT SECTOR IS   */
/* READ INTO MEMORY AND POINTERS ARE UPDATED.           */
/* CALLED BY: GNC                                       */
/*********************************************************/
    STACK=HL; STACK=BC;
    HL=CHAR$PNTR; DE=[MDBUF$MAX];
    IF ([SUB$HL$DE]) PLUS THEN  /* CHAR$PNTR >=MDBUF$MAX */
       DO;
          HL=EOE; DE=HL;
          /* COMPUTE C$SECTOR - EOE AND TEST >0 */
          HL=C$SECTOR,-1;
          IF ([SUB$HL$DE]) PLUS THEN
             DO;  /* NO MORE DISK SPACE */
                A='S'; [WRITE$TERMINAL 'A'];
                A='P'; [WRITE$TERMINAL 'A'];
                CALL [ERROR];
             END;
          /* READ DISK BUFFER */
          BC=(HL=C$SECTOR);
          DE=[MDBUF]; L=1;  /* READ */
          CALL [MINI$DISK];
          IF (A::[TRUE]) ZERO THEN  /* DISK ERROR */
             CALL [ERROR];
          C$SECTOR=(HL=C$SECTOR,+1);
          /* COMPUTE REC$LEN=REC$LEN-(CHAR$PNTR - REC$PNTR) */
          HL=REC$PNTR; DE=HL;
          HL=CHAR$PNTR;
          [SUB$HL$DE]; DE=HL;
          H=0; L=(A=REC$LEN);
          [SUB$HL$DE]; REC$LEN=(A=L);
          /* RESET REC AND CHAR POINTERS */
          HL=CHAR$PNTR; DE=512;
          [SUB$HL$DE];  CHAR$PNTR=HL;
          REC$PNTR=HL;
       END;
    BC=STACK; HL=STACK;
    END CHECK$RD$PNTR;

GNC: PROCEDURE;
/*********************************************************/
/* ROUTINE TO RETURN NEXT CHARACTER. THIS ROUTINE       */
/* ELIMINATES SYCOR INFORMATION FROM THE RECORD AND     */
```

```
/* CONVERTS CHARACTER TO HEX IF VALID VALUE. THE        */
/* FORMAT OF THE SYCOR FILE FOLLOWS:                    */
/* ABCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCBA */
/* WHERE A IS 00H IF THE RECORD IS ACTIVE OR 80H IF     */
/* THE RECORD HAS BEEN DELETED. B CORRESPONDS TO THE    */
/* NUMBER OF C'S (OR BYTES) IN THE RECORD.              */
/*                                                      */
/* OUTPUT: IF HEX CHAR READ                             */
/*            CY=0, A=HEX VALUE                         */
/*         IF NOT A HEX CHAR                            */
/*            CY=1, A=ASCII CHAR                        */
/* CALLED BY: READ,GNB                                  */
/********************************************************/
    STACK=BC; STACK=DE; STACK=HL;
    CALL CHECK$RD$PNTR;  /* INSURE CHARACTER IN MEMORY */
    B=0; C=(A=REC$LEN);
    HL=REC$PNTR; DE=HL; HL=CHAR$PNTR;
    /* IF CHAR$PNTR - REC$PNTR = REC$LEN */
    /*    THEN PROCESS NEW SYCOR RECORD      */
    IF ([SUB$HL$DE]; A=H; A::B) ZERO & (A=L; A::C) ZERO THEN
       DO;  /* PROCESS NEW SYCOR RECORD */
          EXIT=(A=[FALSE]);
          REPEAT;
             HL=CHAR$PNTR;
             B=M(HL); /* SYCOR RECORD EXISTENCE FLAG */
             CHAR$PNTR=(HL=HL+1);
             CALL CHECK$RD$PNTR;
             REC$LEN=(A=M(HL),+1,+1);
             CHAR$PNTR=(HL=HL+1);
             CALL CHECK$RD$PNTR;
             IF (A=<B) CY THEN
                DO;  /* SYCOR RECORD DELETED */
                   B=0; C=(A=REC$LEN);
                   /* CHAR$PNTR=CHAR$PNTR+REC$LEN */
                   CHAR$PNTR=(HL=HL+BC);
                   CALL CHECK$RD$PNTR;
                END
             ELSE
                EXIT=(A=[TRUE]);  /* SYCOR RECORD VALID */
          UNTIL (A=EXIT; A::[TRUE]) ZERO;
          REC$PNTR=HL;  /* REC$PNTR=CHAR$PNTR */
       END;  /* PROCESS NEW SYCOR RECORD */
       HL=CHAR$PNTR; A=M(HL);
       /* IF (CHAR > '9' AND CHAR < 'A') OR */
       /* CHAR < '0' OR CHAR > 'F'          */
       IF (IF (A::3AH) PLUS & (A::41H) MINUS THEN
             CY=1 ELSE CY=0)
             CY \ (A::30H) MINUS \ (A::47H) PLUS THEN
          CY=1  /* NOT A HEX CHAR */
       ELSE
          DO;  /* HEX CHAR - CONVERT */
             A=A-30H;
             IF (A::10) PLUS THEN
                A=A-7;
             CY=0;
          END;
    CHAR$PNTR=(HL=CHAR$PNTR,+1);
    HL=STACK; DE=STACK; BC=STACK;
    END GNC;

GNB: PROCEDURE;
/********************************************************/
/* THE GET NEXT BYTE ROUTINE READS TWO CHARACTERS AND   */
/* FORMS A BYTE. THE CHECKSUM IS ALSO UPDATED IN THIS   */
/* ROUTINE.                                             */
/* INPUT: B - CHECKSUM                                  */
/* OUTPUT: A - BYTE VALUE                               */
/*         B - UPDATED CHECKSUM                         */
/* CALLED BY: READ                                      */
/********************************************************/
    STACK=DE;
    CALL GNC; A=<<A; A=<<A; A=<<A; D=(A=<<A);
    CALL GNC; D=(A=A+D);
```

```
    B=(A=B+D);  /* UPDATE CHECKSUM */
    A=D;
    DE=STACK;
    END GNB;

READ: PROCEDURE;
/*********************************************************/
/* THIS ROUTINE READS DATA FROM A SYCOR FILE ON THE      */
/* MINI-DISK IN INTEL HEX FORMAT AND LOADS MEMORY.       */
/* ONE REQUIRED PARAMETER IN INPUT. THIS PARAMETER       */
/* CORRESPONDS TO THE <N> IN THE SYCOR FILENAME          */
/* .DISK<N>.                                             */
/* CALLED BY: DEBUG$CMD                                  */
/*********************************************************/
    DECLARE (START,DONE) LABEL;
    CALL [GET$PARAM];
    IF !CY \ (A::0) !ZERO THEN
       CALL [ERROR];  /* INPUT ERROR */
    CALL INIT;  /* CHECK FOR DISK VALIDITY */
    /* INITIALIZE VARIABLES */
    C$SECTOR=(HL=BOE);  /* SET CURRENT SECTOR NUMBER */
    CHAR$PNTR=(HL=[MDBUF$MAX]); REC$PNTR=HL;
    REC$LEN=(A=0);
START:
    /* SCAN TO SEMI-COLON */
    DO WHILE (CALL GNC) !CY & (A::3AH) !ZERO;
    END;
    B=0;  /* CLEAR CHECKSUM */
    /* GET RECORD LENGTH */
    CALL GNB; C=A;
    IF (A::0) ZERO THEN
       GOTO DONE; /* 0 LENGTH RECORD - NORMAL TERMINATION */
    /* GET LOAD ADDRESS */
    CALL GNB; H=A; CALL GNB; L=A;
    /* STRIP OFF RECORD TYPE */
    CALL GNB;
    REPEAT;  /* FILL DATA LOCATIONS */
       CALL GNB;
       M(HL)=A;
    UNTIL (HL=HL+1; C=(A=C-1); A::0) ZERO;
    IF (CALL GNB; A=B; A::0) !ZERO THEN
       DO; /* CHECKSUM ERROR */
          A='C'; [WRITE$TERMINAL 'A'];
          A='S'; [WRITE$TERMINAL 'A'];
          CALL [ERROR];
          END;
    /* ADJUST CHAR$PNTR TO POINT TO START OF */
    /* NEXT SYCOR RECORD                     */
    H=0; L=(A=REC$LEN); DE=HL;
    CHAR$PNTR=(HL=REC$PNTR,+DE);
    GOTO START;
DONE:
    END READ;

DISK$IO: PROCEDURE;
/***************************************************/
/* THIS ROUTINE UTILIZES CALLS TO THE MINI$DISK    */
/* PROCEDURE TO READ OR WRITE SPECIFIED NUMBERS    */
/* OF SECTORS BETWEEN DISK AND MEMORY.  THE DMA    */
/* ADDRESS IS SPECIFIED IN THE COMMAND.            */
/* CALLED BY: DEBUG$CMD                            */

/***************************************************/
    DECLARE (INPUT, OUTPUT, START1) LABEL;
    DECLARE (DISK$OP, N$SEC) BYTE INITIAL (0,0);
    INPUT:
        DISK$OP=(A=1);
        GOTO START1;
    OUTPUT:
        DISK$OP=(A=2);
    START1:
    CALL [GET$PARAM];
        IF CY \ (A::0) !ZERO THEN CALL [ERROR];
```

```
        STACK=HL; /* SAVE THE STARTING SECTOR # */
    CALL [GETSPARAM];
        IF CY \ (A::0) !ZERO THEN CALL [ERROR];
        STACK=HL; /* STACK START ADDRESS */
    CALL [GETSPARAM];
        IF !CY \ (A::0) !ZERO THEN CALL [ERROR];
    /* SET # OF SECTORS - SET UP FOR CALL TO MINIDISK */
    N$SEC=(A=L); L=(A=DISK$OP);
    DE=STACK; BC=STACK;
    REPEAT;
        CALL [MINI$DISK];
        IF (A::0) !ZERO THEN CALL [ERROR];
        DE=(HL=200H+DE);  BC=BC+1;  L=(A=DISK$OP);
        N$SEC=(A=N$SEC,-1);  /* READJUST COUNTERS */
    UNTIL (A::0) ZERO;
    END DISK$IO;
```

EOF

```
   /*********************************************************/
   /*  *  *  *  * MTS COMMAND PROCESSOR (MCP)    *  *  *  */
   /*********************************************************/

19FAH:

MCP:  PROCEDURE;

   /*********************************************************/
   /*  MCP IS AN INDEPENDENT MODULE OF THE MICROCOMPUTER
     TIMESHARED SYSTEM (MTS) DEVELOPED FOR THE NPS
     MICROCOMPUTER LABORATORY SYCOR 440 SYSTEM.
     THIS MODULE IS CALLED BY THE MTS MONITOR TO
     PROCESS ANY SYSTEM COMMANDS ENTERED THROUGH THE
     TERMINAL INTERFACE BY THE USER.  MTS COMMANDS ARE
     VALIDATED BY MCP AND THEN SENT TO MTS SERVICE
     CALL CONTROL MODULE FOR FURTHER PROCESSING.
     MCP IS WRITTEN IN PLM FOR TWO REASONS:
          (1) TO UTILITZE A HIGH-LEVEL LANGUAGE TO
              FACILITATE THE DESIGN AND DEBUGGING TASK
              DURING THE DEVELOPMENT OF MCP.
          (2) TO PROVIDE A PLM PROGRAM WHICH ILLUSTRATES
              THE FUNCTION CALL REQUIREMENTS
              FOR ANY USER PROGRAM TO INTERFACE WITH MTS.

     THERE ARE TWO PRIMARY DIFFERENCES BETWEEN THE MCP
     INTERFACE WITH MTS AND A USER PROGRAM/MTS INTERFACE.
          (1) ONE IS THE ENTRY PORT.  THE MTS INTERFACE PORT
              FOR USER PROGRAMS IS 2000H.  THE ENTRY PORT
              FOR MCP IS 1F00H.
          (2) THE OTHER DIFFERENCE IS THAT USER PROGRAMS
              DO NOT HAVE TO BE CONCERNED WITH SAVING
              AND RESTORING THE MTS MONITOR STACKPTR.

     MCP WILL PROCESS THE FOLLOWING SYSTEM COMMANDS
     ENTERED AT THE TERMINAL BY THE USER:

          COMMAND          PARAMETERS

          LOGIN     <DISK NUMBER> /<KEY>
          QUIT           NONE
          ATTACH    <DRIVE LETTER> <DISK NUMBER> /<KEY>
          PROTECT   <DISK NUMBER> /<KEY>
          RESTRICT  <DISK NUMBER> /<KEY>
          UNPROTECT <DISK NUMBER> /<KEY>
          SIZE      <MEMORY SIZE>

     WHERE
          <DRIVE LETTER>- DESIGNATES A VIRTUAL FLOPPY DISK
                          DRIVE TO BE ONE OF THE LETTERS
                          A THRU H.
          <DISK NUMBER> - SPECIFIES A VIRTUAL FLOPPY DISK
                          NUMBER FROM 0-31.
          <DISK NUMBER> - SPECIFIES A DISK NUMBER FROM 0-31.
          /<KEY>        - SPECIFIES A 4 CHARACTER KEY CODE.
          <MEMORY SIZE> - SPECIFIES THE SIZE OF THE USER
                          PROGRAM AREA.
   */
   /*********************************************************/


   /*********************************************************/
   /*  *  *  *  MCP LITERAL AND DATA DECLARATIONS  *  *  */
   /*********************************************************/


   /*  *  *  *  *  MCP LITERAL DECLARATIONS  *  *  *  *  */

DECLARE
    LIT             LITERALLY 'LITERALLY',

    MTS             LIT '1F00H',/* INTERNAL MTS PORT      */
    MTS$CMD$READY   LIT '0F0H' ,
```

```
    INVALID$CMD      LIT '1'   ,

    TRUE             LIT '0FFH',
    FALSE            LIT '0'   ,
    CR               LIT '0DH' ,
    LF               LIT '0AH' ,
    TAB              LIT '9'   ,

    MAX$CBUFF$SIZE   LIT '64'  ,

    READ$MTS$CMD     LIT '-1'  ,  /* INTERNAL MTS CMD   */
    ATTACH           LIT '0'   , /*MTS SYSTEM CMD NUMBERS*/
    MTS$MSG          LIT '1'   ,
    LOGIN            LIT '2'   ,
    PROTECT          LIT '3'   ,
    QUIT             LIT '4'   ,
    RESTRICT         LIT '5'   ,
    SIZE             LIT '6'   ,
    UNPROTECT        LIT '7'   ,
    TERMINAL$STATUS  LIT '8'   ,
    READ$TERMINAL    LIT '9'   ,
    WRITE$TERMINAL   LIT '10'  ;

  /*  *  *  *  *  MCP GLOBAL DECLARATIONS   *  *  *  *  */

DECLARE
    CBUFF (64)      BYTE, /* CMD BUFF FOR MTS COMMAND */
    CBUFF$LENGTH    BYTE, /* NUMBER OF CHARS IN CBUFF */
    CBUFF$PTR       BYTE; /* PTS TO NEXT CHAR IN CBUFF
                             TO BE PROCESSED              */

DECLARE
    CHAR            BYTE, /* USED FOR CHAR MANIPULATION     */
    VALUE (2)       BYTE, /* VECTOR FOR CONVERTING NUMBERS */
    PARAMETERS(6)   BYTE; /*PARAMETERS FOR MTS SYSTEM CALLS*/


  /*****************************************************/
  /*  *  *  *  *  MTS INTERFACE PROCEDURES  *  *  *  *  */
  /*****************************************************/


    /****************************************************/
    /*   MTS1- PROVIDES MTS INTERFACE FOR FUNCTIONS
         WHICH DO NOT REQUIRE A RETURN VALUE.
         'F' CONTAINS THE MTS CMD NUMBER; 'A' CONTAINS
         THE PARAMETER OR ADDRESS TO LIST OF PARAMETERS.
         CALLED BY: ERROR$MSG; SEND$MTS$CMD
    */
MTS1:  PROCEDURE (F,A);
    DECLARE F BYTE, A ADDRESS;
    GO TO MTS;
    END MTS1;

    /***************************************************/
    /*   MTS2- PROVIDES MTS INTERFACE FOR FUNCTIONS
         WHICH REQUIRE A RETURNED VALUE.
         'F' AND 'A' ARE THE SAME AS IN MTS1.
         CALLED BY: READ$CHAR; SEND$MTS$CMD
    */
MTS2:  PROCEDURE (F,A) BYTE;
    DECLARE F BYTE, A ADDRESS;
    GO TO MTS;
    END MTS2;


  /*****************************************************/
  /*  *  *  *  *  *  MCP PRIMITIVE PROCEDURES  *  *  *  */
  /*****************************************************/

READ$CHAR:  PROCEDURE BYTE;
    RETURN MTS2(READ$MTS$CMD, 0);
    END READ$CHAR;
```

```
CBUFF$NOT$EMPTY:  PROCEDURE BYTE;
    RETURN CBUFF$PTR < CBUFF$LENGTH;
    END CBUFF$NOT$EMPTY;


DEBLANK:  PROCEDURE;
    DO WHILE (CBUFF$PTR < CBUFF$LENGTH) AND
             (CBUFF(CBUFF$PTR) = ' ') OR
             (CBUFF(CBUFF$PTR) = TAB);
         CBUFF$PTR = CBUFF$PTR + 1;
         END;
    END DEBLANK;


ERROR$MSG:  PROCEDURE;
    CALL MTS1(MTS$MSG, INVALID$CMD);
    GOTO FINI;
    END ERROR$MSG;


    /*******************************************************/
    /*   FILL$CBUFF - CHECK CURRENT TERMINAL STATUS FOR
         MTS COMMAND. IF NOT, EXIT; OTHERWISE FILL THE
         COMMAND BUFFER WITH THE MTS COMMAND.
         CALLED BY: MCP MAIN CONTROL
    */
FILL$CBUFF:  PROCEDURE;
    IF MTS2(TERMINAL$STATUS,0)=MTS$CMD$READY THEN
         DO;
         CBUFF$PTR, CBUFF$LENGTH = 0;
         DO WHILE (CBUFF$LENGTH <= MAX$CBUFF$SIZE) AND
                  ((CBUFF(CBUFF$LENGTH):=READ$CHAR)<>CR);
              CBUFF$LENGTH = CBUFF$LENGTH+1;
              END;
         END;
    END FILL$CBUFF;


INITIALIZE:  PROCEDURE;
    DECLARE I BYTE;
    DO I=0 TO 5;
    PARAMETERS(I) = 0FFH;
         END;
    VALUE(0), VALUE(1) = 0;
    END INITIALIZE;


    /*******************************************************/
    */
LETTER:  PROCEDURE (C) BYTE;
    DECLARE C BYTE;
    RETURN C >= 'A' AND C <= 'Z';
    END LETTER;


    /*******************************************************/
    /*   NUMBER - RETURN TRUE IF 'C' IS A NUMBER.
         CALLED BY: GET$NUMBER; GET$PARAMETERS;
                    ATTACH$CMD; SIZE$CMD
    */
NUMBER:  PROCEDURE (C) BYTE;
    DECLARE C BYTE;
    RETURN C >= '0' AND C <= '9';
    END NUMBER;


    /*******************************************************/
    /*   READ$CMD$LINE - READS CHAR FROM COMMAND BUFFER;
         CONVERTS LETTERS FROM LOWER TO UPPER CASE,
         IF REQUIRED.
         CALLED BY: GET$KEY; GET$NUMBER; GET$PARAMETERS;
```

```
                    ATTACH$CMD; SIZE$CMD; MCP MAIN CONTROL
    */
READ$CMD$LINE:  PROCEDURE BYTE;
    DECLARE C BYTE;
    IF (C:=CBUFF(CBUFF$PTR)) >= 61H /* LOWER CASE A      */
       AND C <= 7AH THEN          /*  LOWER CASE Z     */
         C = C AND 5FH;    /* CONVERT TO UPPER CASE      */
    CBUFF$PTR = CBUFF$PTR+1;
    RETURN C;
    END READ$CMD$LINE;


    /****************************************************/
    /*   SCAN$TO$BLANK - SCAN TO NEXT BLANK OR TAB CHAR.
         CALLED BY: MCP MAIN CONTROL
    */
SCAN$TO$BLANK:  PROCEDURE;
    DO WHILE (CBUFF$PTR<CBUFF$LENGTH) AND
             (CBUFF(CBUFF$PTR)<>' ') AND
             (CBUFF(CBUFF$PTR)<>TAB);
         CBUFF$PTR = CBUFF$PTR+1;
         END;
    END SCAN$TO$BLANK;


    /****************************************************/
    /*   SEND$MTS$CMD - 'CMD' CONTAINS THE MTS CMD NUMBER
         AND 'PARAMETER' CONTAINS THE ACTUAL PARAMETER
         OR THE ADDRESS OF THE ACTUAL PARAMETER LIST.
         TWO MTS SYSTEM CALLS ARE MADE:
         (1) MTS2 - PROCESS SYSTEM CMD, WHICH RETURNS
                    A RESPONSE.
         (2) MTS1 - DISPLAY RESPONSE AT USER TERMINAL.
    */
SEND$MTS$CMD:  PROCEDURE (CMD,PARAMETER);
    DECLARE CMD BYTE, PARAMETER ADDRESS;
    CALL MTS1(MTS$MSG, MTS2(CMD,PARAMETER));
    END SEND$MTS$CMD;


   /****************************************************/
   /*  *  *  *  *  UTILITY PROCEDURES  *  *  *  *  *  *  */
   /****************************************************/


    /****************************************************/
    /*   CONVERT$VALUE - CONVERT ASCII CODE IN 'VALUE'
         TO APPROPRIATE BINARY VALUE.  THE RANGE OF VALUES
         TO BE CONVERTED ARE FROM 0-31.
         CALLED BY: GET$NUMBER;
    */
CONVERT$VALUE:  PROCEDURE BYTE;
    IF VALUE(1) = 0 THEN /*ONLY A SINGLE DIGIT TO CONVERT*/
         RETURN VALUE(0)-30H;
    ELSE                 /* TWO DIGITS TO CONVERT         */
         RETURN ((VALUE(0)-30H)*10+(VALUE(1)-30H));
    END CONVERT$VALUE;


    /****************************************************/
    /*   GET$KEY - GET THE <KEY> PARAMETER, IF ENTERED;
         STORE KEY IN THE PARAMETER LIST STARTING AT I.
         CALLED BY: ATTACH$CMD; GET$PARAMETERS;
    */
GET$KEY:  PROCEDURE (I);
    DECLARE I BYTE;
    IF CBUFF$NOT$EMPTY THEN   /*NEXT CHAR MUST BE '/'     */
         DO;
         IF (CHAR:=READ$CMD$LINE) = '/' THEN
              DO WHILE (I<6) AND CBUFF$NOT$EMPTY;
                   PARAMETERS(I)=READ$CMD$LINE;
                   I=I+1;
              END;
```

```
        ELSE                    /* '/' IS USED TO INDICATE */
            CALL ERROR$MSG; /* <KEY> AND IS REQUIRED.  */
        END;
    END GET$KEY;


    /*****************************************************/
    /*  GET$NUMBER - GET <DISK NUMBER> PARAMETER AND
        CONVERT IT FROM ASCII TO BINARY. STORE THE
        RESULT IN PARAMETER LIST AT 'I'. UPON ENTRY,
        'CHAR' HOLDS THE FIRST DIGIT.
        CALLED BY: ATTACH$CMD; GET$PARAMETERS;
    */
GET$NUMBER:  PROCEDURE (I);
    DECLARE I BYTE;
    VALUE(0) = CHAR;
    IF CBUFF$NOT$EMPTY AND
       NUMBER(CHAR:=READ$CMD$LINE)
       THEN                 /*  TWO DIGIT DISK NUMBER      */
         VALUE(1) = CHAR;
    PARAMETERS(I) = CONVERT$VALUE;
    END GET$NUMBER;


    /*****************************************************/
    /*  GET$PARAMETERS - USED TO GET THE <DISK NUMBER>
        AND <KEY> PARAMETERS. GENERATES ERROR MSG IF
        NEXT CHAR IS NOT A NUMBER.
        CALLED BY: LOGIN$CMD; GET$REQUIRED$PARAMETERS;
    */
GET$PARAMETERS:  PROCEDURE;
    IF NUMBER(CHAR:=READ$CMD$LINE) THEN
        DO;
        CALL GET$NUMBER(0);
        CALL DEBLANK;
        CALL GET$KEY(1);
        END;
    ELSE
        CALL ERROR$MSG;
    END GET$PARAMETERS;


    /*****************************************************/
    /*  GET$REQUIRED$PARAMETERS - GETS THE <DISK NUMBER>
        AND <KEY> PARAMETERS FOR THOSE SYSTEM CMDS FOR
        WHICH THESE PARAMETERS ARE REQUIRED (NOT
        OPTIONAL). GENERATES ERROR MSG IF PARAMETERS
        ARE NOT THERE.
        CALLED BY:PROTECT$CMD; RESTRICT$CMD; UNPROTECT$CMD
    */
ET$REQUIRED$PARAMETERS:  PROCEDURE;
    IF CBUFF$NOT$EMPTY THEN
        CALL GET$PARAMETERS;
    ELSE
        CALL ERROR$MSG;
    END GET$REQUIRED$PARAMETERS;


 /******************************************************/
 /*  *  *  *  *  *  SYSTEM CMD PROCEDURES  *  *  *  *  */
 /******************************************************/


    /*****************************************************/
    /*  ATTACH$CMD -
        FORM:  ATTACH <DRIVE LTR> <DISK NR> /<KEY>
        ALL THE PARAMETERS ARE OPTIONAL, HOWEVER
        <KEY> CAN NOT APPEAR WITHOUT IT'S ASSOCIATED
        <DISK NUMBER>. WHEN PARAMETERS ARE ENTERED
        THEY MUST BE IN THE ORDER INDICATED.
        CALLED BY: MCP MAIN CONTROL
    */
A]ACH$CMD:  PROCEDURE;
```

```
    IF CBUFF$NOT$EMPTY THEN
         DO;
         CHAR = READ$CMD$LINE;
         IF LETTER(CHAR) THEN         /* <DRIVE LETTER>       */
              DO;
              PARAMETERS(0)=CHAR-41H;/*CONVERT TO BINARY*/
              CALL DEBLANK;
              IF CBUFF$NOT$EMPTY THEN /* MORE PARAMETERS */
                   CHAR = READ$CMD$LINE;
              END;
         IF NUMBER(CHAR) THEN         /* <DISK NUMBER>        */
              DO;
              CALL GET$NUMBER(1);
              CALL DEBLANK;
              CALL GET$KEY(2);        /*  <KEY>               */
              CALL DEBLANK;
              END;
         END;
    IF CBUFF$NOT$EMPTY THEN
         CALL ERROR$MSG;
    ELSE
         CALL SEND$MTS$CMD(ATTACH,.PARAMETERS);
    END ATTACH$CMD;


    /*****************************************************/
    /*   LOGIN$CMD -
         FORM: LOGIN <DISK NUMBER> /<KEY>
         THE PARAMETERS ARE OPTIONAL BUT <KEY> CAN NOT
         APPEAR WITHOUT <DISK NUMBER>.
         CALLED BY: MCP MAIN CONTROL
    */
LOGIN$CMD:  PROCEDURE;
    IF CBUFF$NOT$EMPTY THEN
         CALL GET$PARAMETERS;
    CALL SEND$MTS$CMD(LOGIN,.PARAMETERS);
    END LOGIN$CMD;


    /*****************************************************/
    /*   PROTECT$CMD -
         FORM: PROTECT <DISK NUMBER> /<KEY>
         THE PARAMETERS ARE REQUIRED.
         CALLED BY: MCP MAIN CONTROL
    */
PROTECT$CMD:  PROCEDURE;
    CALL GET$REQUIRED$PARAMETERS;
    CALL SEND$MTS$CMD(PROTECT, .PARAMETERS);
    END PROTECT$CMD;


    /*****************************************************/
    /*   QUIT$CMD - FORM:  QUIT
         NO PARAMETERS.
         CALLED BY: MCP MAIN CONTROL
    */
QUIT$CMD:  PROCEDURE;
    CALL SEND$MTS$CMD(QUIT,0);
    END QUIT$CMD;


    /*****************************************************/
    /*   RESTRICT$CMD -
         FORM: RESTRICT <DISK NUMBER> /<KEY>
         THE PARAMETERS ARE REQUIRED.
         CALLED BY: MCP MAIN CONTROL
    */
RESTRICT$CMD:  PROCEDURE;
    CALL GET$REQUIRED$PARAMETERS;
    CALL SEND$MTS$CMD(RESTRICT,.PARAMETERS);
    END RESTRICT$CMD;
```

```
    /*******************************************************/
    /*   UNPROTECT$CMD -
         FORM: UNPROTECT <DISK NUMBER> /<KEY>
         THE PARAMETERS ARE REQUIRED.
         CALLED BY: MCP MAIN CONTROL
    */
UNPROTECT$CMD:  PROCEDURE;
    CALL GET$REQUIRED$PARAMETERS;
    CALL SEND$MTS$CMD(UNPROTECT,.PARAMETERS);
    END UNPROTECT$CMD;


    /*******************************************************/
    /*   SIZE$CMD -
         FORM: SIZE <MEMORY SIZE>
         THE PARAMETER IS REQUIRED.
         CALLED BY: MCP MAIN CONTROL
    */
SIZE$CMD:  PROCEDURE;
    IF CBUFF$NOT$EMPTY THEN
         DO;
         IF NUMBER(CHAR:=READ$CMD$LINE) THEN
              DO;          /* GET MEMORY SIZE PARAMETER  */
              CALL GET$NUMBER(0);
              CALL SEND$MTS$CMD(SIZE, PARAMETERS(0));
              END;
         ELSE              /* PARAMETER MUST BE NUMBER        */
              CALL ERROR$MSG; /* SPECIFYING MEMORY SIZE  */
         END;
    ELSE                   /* CBUFF EMPTY - ERROR             */
         CALL ERROR$MSG; /*          PARAMETER REQUIRED  */
    END SIZE$CMD;


  /*******************************************************/
  /*  *  *  MTS COMMAND PROCESSOR (MCP) MAIN CONTROL *  */
  /*  *  *  CALLED BY:  MTS MONITOR MODULE          *  */
  /*******************************************************/


DECLARE STACK (20) ADDRESS,OLDSP ADDRESS;

OLDSP = STACKPTR;          /* SAVE MTS STACK POINTER        */
STACKPTR = .STACK(LENGTH(STACK));  /* SETUP MCP STACKPTR */

CALL INITIALIZE;           /* INITIALIZE DATA STRUCTURES    */
CALL FILL$CBUFF;           /* GET MTS COMMAND               */
CALL DEBLANK;              /* SCAN TO FIRST NONBLANK CHAR   */
IF CBUFF$NOT$EMPTY THEN    /* PROCESS CMD BUFFER            */
    DO;
    CHAR = READ$CMD$LINE;/* GET FIRST LETTER OF CMD       */
    CALL SCAN$TO$BLANK;  /* SCAN TO NEXT BLANK, BECAUSE
                            ONLY THE FIRST LETTER IS USED
                            TO DETERMINE THE CMD          */
    CALL DEBLANK;

    IF CHAR = 'A' THEN                         /* ATTACH   */
         CALL ATTACH$CMD;
    ELSE
         DO;
         IF CHAR = 'L' THEN                    /* LOGIN    */
              CALL LOGIN$CMD;
         ELSE
              DO;
              IF CHAR = 'P' THEN               /* PROTECT  */
                   CALL PROTECT$CMD;
              ELSE
                   DO;
                   IF CHAR = 'Q' THEN          /* QUIT     */
                        CALL QUIT$CMD;
                   ELSE
                        DO;
                        IF CHAR = 'R' THEN   /* RESTRICT */
```

```
                                    CALL RESTRICT$CMD;
                              ELSE
                                    DO;
                                    IF CHAR='S' THEN /* SIZE     */
                                          CALL SIZE$CMD;
                                    ELSE
                                      DO;
                                      IF CHAR='U' THEN/*UNPROTECT*/
                                            CALL UNPROTECT$CMD;
                                      ELSE
                                            CALL ERROR$MSG;
                                      END;
                                    END;
                              END;
                        END;
                  END;
            END;
      END;

FINI:
STACKPTR = OLDSP;  /* RESTORE MTS MONITOR STACKPTR       */

END MCP;


EOF
```

LIST OF REFERENCES

1. Bullock, D. R. and Brown, K. J., A Shared Environment for Microcomputer System Development, M.S. Thesis, Naval Postgraduate School, Monterey, CA, March, 1977.

2. Digital Research, An Introduction to CP/M Features and Facilities, Pacific Grove, CA 93950, 1976.

3. Digital Research, CP/M Dynamic Debugging Tool (DDT) User's Guide, Pacific Grove, CA 93950, 1976.

4. Digital Research, CP/M Interface Guide, Pacific Grove, CA 93950, 1976.

5. Digital Research, CP/M System Alteration Guide, Pacific Grove, CA 93950, 1976.

6. Digital Research, ED: Context Editor for the CP/M Disk System User's Manual, Pacific Grove, CA 93950, 1975.

7. Digital Research, CP/M Assembler (ASM) User's Guide, Pacific Grove, CA 93950, 1976.

8. Intel Corp., 8080 Assembly Language Programming Manual, Santa Clara, CA 95051, 1976.

9. Intel Corp., 8008 and 8080 PL/M Programming Manual, Santa Clara, CA 95051, 1976.

10. Madnick, Stuart E. and Donovan, John J., Operating Systems, McGraw-Hill, 1974.

11. Pedroso, L. R. B., ML80: A Structured Machine-Oriented Microcomputer Programming Language, M.S. Thesis, Naval Postgraduate School, Monterey, CA, December 1975.

12. Sycor Inc., Model 340/340D Intelligent Communications Terminal Operator's Manual, Nr TD34003-275, Ann Arbor, Michigan, October 1974.

13. Sycor Inc., User's Guide to the 440 System, Nr UG4400-2, Ann Arbor, Michigan, November 1976.

14. Sycor Inc., 8080 Debug User's Guide, Spec. 950706, Rev B, Ann Arbor, Michigan, October 1975.

15. Intel Corp., ISIS-II Systems User's Guide, Santa Clara, CA 95051, 1976.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Assoc Professor Gary A. Kildall, Code 52Kd Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	LCDR Stephen T. Holl, USN, Code 52Hl Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	Microcomputer Laboratory, Code 52ec Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
7.	LT Stephen J. Carro, USN 1105 Spruance Road, Monterey, California 93940	1
8.	Mr Barry L. Knouse Naval Air Development Center Code 553 Warminster, Pennsylvania 18974	1

9 OCT 81 S10591

172323

Thesis
C27215
c.1

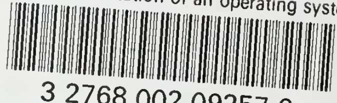
Carro
The implementation of an operating system for a shared microcomputer environment.

9 OCT 81 S10591

172323

Thesis
C27215
c.1

Carro
The implementation of an operating system for a shared microcomputer environment.


thesC27215
The implementation of an operating syste
3 2768 002 09257 9
DUDLEY KNOX LIBRARY