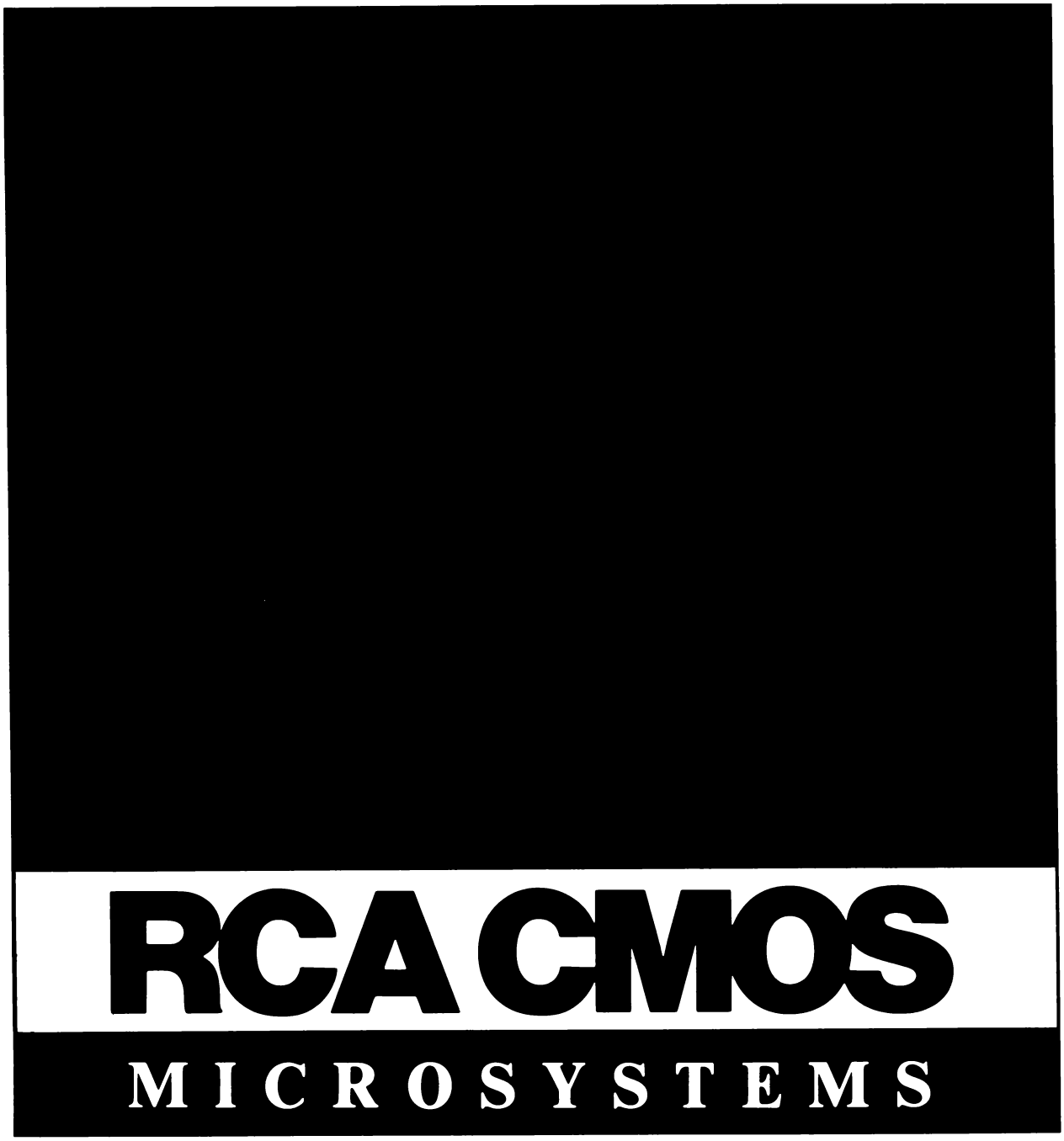
RCA Solid State



User Manual for the RCA MicroDisk Development System MS2000

MPM-241 **Suggested Price $5.00**

User Manual for the RCA MicroDisk Development System MS2000

RCA Solid State
Somerville, NJ • Brussels • Paris • London
Hamburg • Sao Paulo • Hong Kong

CLASS A
RADIO INTERFERENCE WARNING

This equipment complies with the requirements in Part 15 of FCC Rules for a Class A computing device. Operation of this equipment in a residential area may cause unacceptable interference to radio and TV reception requiring the operator to take whatever steps are necessary to correct the interference.

2488834-1

The software described in this manual is copyrighted by RCA Corporation.

Information furnished by RCA is believed to be accurate and reliable. However, no responsibility is assumed by RCA for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of RCA.

Tradmark(s)® Registered
Marca(s) Registrada(s)

Printed in USA 5-84

Foreword

The RCA MicroDisk Development System MS2000 is a microprocessor computer system designed to facilitate the development of hardware and software for applications based on the RCA 1800 series of CMOS microprocessors. It utilizes 3-½ inch, high-density microfloppy disk drives. The disks provide 645 kilobytes of on-line mass memory storage. Featuring higher speeds than its predecessors, the MS2000, with its new DMA controller, has reduced system load time to 0.6 second.

The MicroDisk Development System is contained in a 20-slot Microboard Industrial Chassis containing not only the four Microboards provided, but also the power supply and the complete Dual Microfloppy Disk Drives. The chassis provides four additional spare slots for expansion and enhancements with any of the extensive line of RCA Microboards.

The memory includes 632 kilobytes of RAM, 2 kilobytes of ROM, and 645 kilobytes of on-line mass memory storage on microfloppy disks. Software provided includes an augmented resident monitor program UT71 and the MicroDOS operating system. MicroDOS includes an Editor and a MacroAssembler ASM8 that operates not only with all the RCA CMOS Microprocessors CDP1802A, CDP1805AC, CDP1806C, and CDP1806AC, but with RCA Microprocessors to be added to the expanding line.

Conversion programs are included that provide transportability of source code from all other RCA Development Systems to the MS2000.

Optional add-ons include a PROM Programmer package, BASIC1, BASIC2, the CDP18S040 CRT Terminal providing full-screen editing, and the MS3001 MicroEmulator.

This Manual describes in detail the hardware structure and the software features and commands of the MicroDisk Development System MS2000. The user should also refer to the *User Manual for the CDP1802 Microprocessor,* MPM-201, for a detailed description of the instruction set and the architecture of the CDP1802 CMOS Microprocessor.

CONTENTS

1. System Structure and Set-up

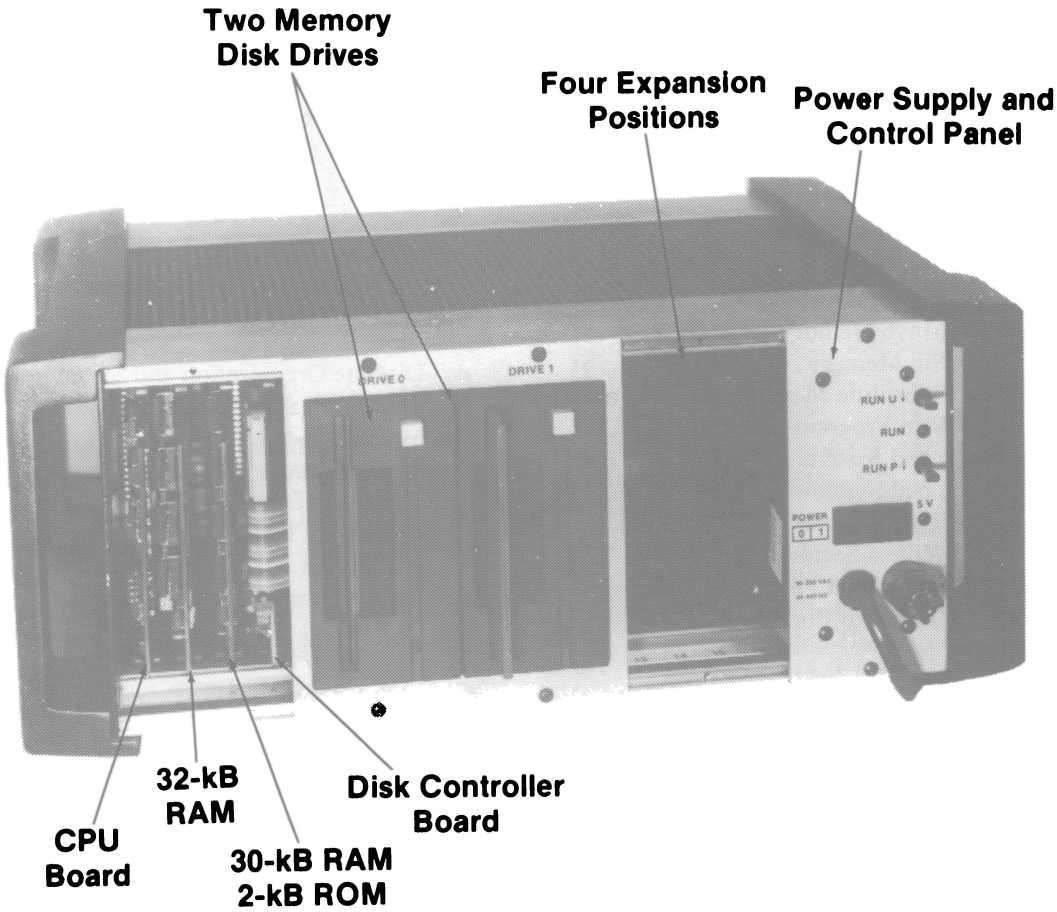


Fig. 1 - MS2000 chassis with two front covers removed to show typical module locations.

One of the features of the MicroDisk Development System MS2000 is its modular construction. Fig. 1 shows an arrangement of the modules that provides good mechanical and electrical balance. The modules that make up the MS2000 include:

1. 20-Slot Microboard Industrial Chassis with Backplane
2. CMOS Microboard Computer (CPU)
3. Microboard Memory Module with 32 Kilobytes of RAM
4. Microboard Memory Module with 30 Kilobytes of RAM and 2 Kilobytes of ROM
5. Microfloppy Disk Controller
6. Dual Disk Drive Module
7. Power Supply Module

Chassis

The chassis supplied with the MS2000 is a 20-slot customized MSI8820 Industrial Chassis. It includes an integral card rack, backplane, and case. The top and bottom covers are perforated and removable. The front and back covers are removable as are the side panels and end bezels.

The backplane is a standard Microboard universal backplane in which any module may occupy any position. To prevent magnetic interference between the MSIM40 power supply and the MSIM50 Disk Drives, always mount the modules with at least four card slots between them. Table I shows the backplane signals and their pin assignments.

The signal naming convention is to give each signal an

Table I—Pin Terminals and Signals for the RCA Microboard Universal Backplane.

Wire Side				Component Side			
Pin	Mnemonic	Signal Flow	Description	Pin	Mnemonic	Signal Flow	Description
A	TPA-P	Out	System Timing Pulse 1	1	DMAI-N	In	DMA Input Request
B	TPB-P	Out	System Timing Pulse 2	2	DMAO-N	In	DMA Output
C	DB0-P	In/Out	Data Bus	3	RNU-P	—	Run Utility Request
D	DB1-P	In/Out	Data Bus	4	INT—N	In	Interrupt Request
E	DB2-P	In/Out	Data Bus	5	MRD-N	Out	Memory Read
F	DB3-P	In/Out	Data Bus	6	Q-P	Out	Programmed Output Latch
H	DB4-P	In/Out	Data Bus	7	SC0-P	Out	State Code
J	DB5-P	In/Out	Data Bus	8	SC1-P	Out	State Code
K	DB6-P	In/Out	Data Bus	9	CLEAR-N	In	Clear-Mode Request
L	DB7-P	In/Out	Data Bus	10	WAIT-N	In	Wait-Mode Request
M	A0-P	Out	Multiplexed Address Bus	11	-5/-15V	—	Auxiliary Power
N	A1-P	Out	Multiplexed Address Bus	12	SPARE	—	Not Assigned
P	A2-P	Out	Multiplexed Address Bus	13	CLOCK OUT	Out	Clock from CPU Osc.
R	A3-P	Out	Multiplexed Address Bus	14	N0-P	Out	I/O Primary Address
S	A4-P	Out	Multiplexed Address Bus	15	N1-P	Out	I/O Primary Address
T	A5-P	Out	Multiplexed Address Bus	16	N2-P	Out	I/O Primary Address
U	A6-P	Out	Multiplexed Address Bus	17	EF1-N	In	External Flag
V	A7-P	Out	Multiplexed Address Bus	18	EF2-N	In	External Flag
W	MWR-N	Out	Memory Write Pulse	19	EF3-N	In	External Flag
X	EF4-N	In	External Flag	20	+12V/+15V	—	Auxiliary Power
Y	+5V		+5V dc	21	+5V		+5V dc
Z	GND		Digital Ground	22	GND		Digital Ground

alphanumeric name descriptive of its major logic function, followed by either -N or -P. The -N means that the named function is true or asserted when the voltage on that particular wire is at ground. The -P means that the named function is true when the voltage is at +5 volts. Thus, a signal NAME-N, after passing through a logic inverter, becomes NAME-P, and vice versa.

The user may wish to rearrange the position of the existing modules when adding expansion modules. For example, if a UART card or a Modem card is added, the two memory cards can be moved to slots 13 through 16 to place the serial-interface card near the left side for ease of cable entry. Alternatively, the cable may be passed under the disk-drive assembly at the front, top, or bottom and the serial card placed in slots 13 through 16. There is sufficient space to pass a 34-wire flat cable (wider cables may be folded). The size of the connector needed with the wider cables will require that the disk module be pulled part way out while placing the cable.

When using the PROM Programmer CDP18S680, the left side panel may be removed and the Programmer placed in slot 1 for access through the left-hand end bezel.

Always allow clearance for air circulation at the top and bottom of the chassis. Overheating and drive or supply failure could result otherwise.

Microboard Computer

The Microboard Computer supplied as the CPU of the system is a variant of the CDP18S605 Microboard Computer. The on-board memory has been left out because the system memory is wholly contained in the two memory Microboards. As a result, the CDP1802A Microprocessor and the CDP1854A UART are the main functional units. The UART provides the serial data path to an external data terminal through an RS232C interface. The baud rate is selectable by the setting of a DIP switch on the CPU Microboard. Baud rates from 50 to 19,200 are available. Table II is a baud rate selection chart showing the position of each of the four rockers of switch S1 for each output baud rate available.

Microboard Memories

Both memory Microboards supplied with the MS2000 are made from the CDP18S628. One is populated with 32 kilobytes of RAM and occupies memory space from 0000H through 7FFFH (H indicates hexadecimal notation). The other is populated with 30 kilobytes of RAM and 2 kilobytes of ROM. The ROM contains the monitor program UT71. The ROM occupies memory space

Table II—Selection Chart Showing Rocker Positions for Each Baud Rate Available on the CPU Board.

Switch S1				Output Rate Baud*
4	3	2	1	
C	C	C	C	19200
C	C	O	C	50
C	C	O	O	75
C	O	C	C	134.5
C	O	C	O	200
C	O	O	C	600
C	O	O	O	2400
O	C	C	C	9600
O	C	C	O	4800
O	C	O	C	1800
O	C	O	O	1200
O	O	C	C	2400
O	O	C	O	300
O	O	O	C	150
O	O	O	O	110

*Actual input to UART is 16 times the indicated output rate, assuming a clock frequency of 2.4578 MHz. O = open; C = clothes.

8000H through 87FFH, and the RAM 8800H through FFFFH.

Microboard Disk Controller

The Microboard Disk Controller CDP18S651 provides the I/O interface between the system software and logic and the two disk drives. Instruction and status data are transferred by output and input commmands; bit data are transferred by Direct Memory Access (DMA). The logic to control the DMA process is built into the disk controller Microboard to interface with the on-chip DMA controller of the CDP1802A on the CPU Microboard. At the end of a DMA transfer, external flag EF3 is used to signal the completion to the software.

The monitor program UT71 contains the I/O driver routines for performing all the commands for the disk operating system (MicroDOS). The disk controller can perform the following functions:

1. Seek a track
2. Format a track
3. Write a sector
4. Read a sector
5. Read multiple sectors
6. Write multiple sectors
7. CRC READ (Read without data transfer but with error checking).
8. Recalibrate (Return heads to home position on track 00).

The disk controller is capable of a variety of formats. Appendix A - **Diskette Organization and Structure** shows the format and disk organization used by the MS2000 MicroDisk Development System.

Dual Disk Drives

The two MicroDisk drives are contained in the MSIM50 module. The module occupies eight slots in the 20-slot chassis. An edge connector picks up power from the backplane, and power-conditioning circuits then provide +5 and +12 volts to the two disk drives. The signal cable is a "daisy chain" configuration using a 26-wire flat cable. The controller end of this cable is a 50-pin connector mating with the CDP18S651 Microboard Controller. The controller is located immediately to the left of the disk drive module in the chassis. Be careful that the cable doesn't "push" on the cover of Drive 0: disk errors will result.

The drives are labeled 0 and 1, corresponding to the drive number used in MicroDOS commands. Drive 0 is the left drive.

The mating 3½-inch diskette has a hard cover with a sliding cover over the head access window. As supplied, the diskettes are not write protected. Activate this feature by breaking out the protect tab, rotating it 90° counterclockwise, and reinserting it. Slide the tab outward for write protect and inward for write enable.

Always mount the MSIM50 at least four card slots away from the MSIM40.

Power Supply

The MSIM40 Power Supply Module plugs into the system chassis and occupies four slots. The edge connector supplies +5, +15, and -15 volts to the system backplane and interfaces the control logic to the system.

An AC input cord, fuseholder, power on-off switch, and power-on indicator (+5 volt LED) are on the front panel. In addition to the power functions, the front panel provides two system control switches and a running indicator.

The two control switches are momentary-action, double-throw types having a center-off position. The RUN UTILITY (RNU) switch, when pressed down, causes a system reset followed by a start at address 8000H, the beginning of the monitor program UT71. The RUN PROGRAM (RNP) switch, when pressed down, causes a system reset followed by a start at address 0000H, where a user program may have been stored in RAM. If either switch is pressed upward, a system reset is generated and latched until either switch is pressed down. The indicator LED labeled RUN is lighted during program execution and extinguished when an IDLE instruction, a WAIT condition, a

RESET condition,or any malfunction preventing normal fetching of instructions is encountered.

System Setup

As the first step in system setup, remove the chassis from the carton and place it on a table on its four rubber feet. Using a No. 1 Phillips screw-driver, remove the two screws from the left-most front cover (the one with the "RCA" on it). Remove the cardboard spacer that held the boards in place during shipment. Remove the left-most board (the CPU board) by lifting up on the black card extractor on the top of the board. Push the card extractor down and carefully remove the CPU board. NOTE: Handle the board on the edges only since the CMOS parts on it are sensitive to static electricity. Locate the red four-position baud-rate switch and set the baud rate corresponding to your terminal, as given in Table II. In this table, C means on, O means off. Now reseat the other three boards by lifting up on their extractor. Then push it down and firmly press the boards back into place. Any of the boards may be removed for your inspection but remember to be careful in handling them; and make sure that they are firmly reseated. Finally, replace the CPU board in the left card slot. NOTE: Make sure the component side of the board faces left.

The 10-pin connector on the top edge of the board is the RS-232 terminal connector. Remove the black cable from the parts box, push the 10-pin end of the cable into the back of the chassis between the left rear handle and the chassis body. Then feed it into the slot in the forward part of the plate on the left side next to the CPU board. Finally, place the 10-pin connector over its mating pins on the CPU board, being careful to align the plugged hole with the position of the missing pin. Now connect the other end with the 25-pin D connector, to your terminal. If the sex of the connector is incorrect for your terminal, use the "gender bender" included in the parts box.

Next plug in the computer system and terminal; turn on the terminal, then the computer system. The red "5V" light on the right panel indicates the presence of the +5 volts DC.

Monitor Program Check

With the +5 volts available, the red "RUN" light will come on and an asterisk and UT71 version number will be displayed on the terminal. The asterisk is the prompt for the UT71 Monitor program. (If no asterisk appears, try restarting the monitor program by depressing and releasing the RUN U toggle switch.)

Now type T (CR)

where (CR) means carriage return. The system should respond with

```
MEMORY OK
*
```

The "T" command does a checksum of the Monitor ROM, and does a read-write test on all RAM (RAM is left filled with "AAs").

Now type

```
D8000 20 (CR)
```

The system will respond with

```
8000 7100 F880 B0F8 8CB1 F81F
A1F1 21F8 D073
8010 81F6 CFF9 10FC 8151 F33A
26D1 7381 FF03
*
```

The monitor command "D" displays the contents of memory at the terminal. The command displays the 20 hex (32 decimal) bytes of data starting at location 8000 on the terminal, then returns the prompt. Since terminal communication has been established, the front cover, removed earlier, can be replaced.

Disk Operation Check

The system disks can now be used. Take the blue-plastic-enclosed 3½-inch diskette with the white stick-on label from the parts box. This diskette contains the MicroDOS Operating System, some utility programs, and the Editor and Assembler.

Check to see if this disk has been "write-protected" to prevent data being inadvertently written to it, possibly destroying existing programs. To do this, find the small rectangular cutout in the corner of the back of the diskette, the side with the round metal hub in the center. If the removable tab is either missing or has slid against the outside edge of the cutout, the disk is write protected. If the disk has not been write-protected, you must complete the procedure described in the next paragraph.

Carefully pry up the tab and break it loose. Turn the tab 90° from its original position. On one of the short ends, there is a small protrusion. This will line up with the depression in the side of the slot from which the tab was removed. Carefully insert the tab in the slot, aligning the protrusion on the tab with the depression on the side of the slot, and snap the tab in place. When properly inserted, the tab will slide back and forth in the slot without coming out. Slide the tab towards the closest edge of the diskette. This will write-protect it. You can

un-write-protect a disk with a missing tab by covering the slot with tape.

Turn the diskette over and slide the metal protector so that the oval cutout is in the center of the diskette in line with the load access hole in the blue plastic. The recording media can be seen through this hole. Now insert the diskette into the left disk drive, the one marked "DRIVE O." Orient the diskette so that the metal hub is towards the right (away from the CPU board) and the edge with the head access hole fits into the disk drive slot first. Push the diskette all the way into the drive until it clicks into place and the red light on the drive blinks on then off. The diskette will not latch if improperly oriented. This completes the loading of the diskette.

Auto-shutter diskettes, mounted in drives so-equipped, will open and close the cover automatically.

Now load the disk operating system. Type "L", and the system will load the 12 kilobytes of operating system into memory. About 0.6 second after typing L, the MicroDOS prompt is issued:

(C) Copyright 1982 RCA Corporation
MicroDOS X.X

The ">" sign is the MicroDOS prompt. The X.X will be two digits, the revision number of the diskette (e.g., 0.0).

Now type DIR;S (CR) . This entry will run the disk directory program, which will display the name of the diskette and an alphabetical listing of all the files on the disk.

Next type HELP (CR) and follow the instructions given you on the first screen. The HELP utility gives a brief description and format of each of the MicroDOS utilities.

As a first use for the system prepare a second diskette in the parts box for use. This diskette must be formatted and initialized for MicroDOS;this is done by using FRMT and SYSGEN. Place this diskette in drive 1 in the same manner as described above for the system diskette, but don't write-protect it. Type FRMT (CR) and follow the instructions. When this task is complete, type SYSGEN;E (CR) and follow those instructions. You will then have created a duplicate of the system diskette. The original can be removed and set aside for safe keeping.

This description demonstrates only a very small part of the system capability. Refer to the remainder of this manual for descriptions of the other utilities and the Editor and Assembler.

2. Understanding MicroDOS

Introduction

The Microboard Disk Operating System (MicroDOS) associated with the MicroDisk Development System MS2000 is a powerful and easy-to-use tool for software development. It is an interactive mass-memory storage system capable of dynamic file operation and management. Its commands, obtained via the system console, reference files stored on the diskette. By means of its dynamic operating system, MicroDOS keeps track of changes in file size during software development and allocates disk space as needed. Disk space not needed by a file is freed and made available for use by a different file. The file operating system can have multiple input and output files open at the same time and can thereby provide the user with considerable design flexibility. The operating system also provides a set of functions that can be called by a user program to perform utility operations such as open files, close files, and the like.

MicroDOS System Ingredients

Use of the MicroBoard Disk Operating System (MicroDOS) requires a MicroDisk Development System MS2000. The software needed for MicroDOS operation includes the UT71 Utility Program, provided on ROM, and the programs provided on the MicroDOS System Diskette. These programs include:

On Disk:

1. MicroDOS Operating System (OP. SYS)
2. MicroDOS System Commands (CDSBIN, COPY, DEL, DIR, FREE, MERGE, PRINT, RENAME, SUBMIT, SYSGEN, U, VERIFY)
3. MicroDOS Macro Disk Assembler (ASM8)
4. MicroDOS Disk Editor (EDIT)
5. Memory Save Program (MEM)
6. Diskette File Examination and Modify Program (EXAM)
7. Diskette Diagnostic Program (DIAG)
8. ASM4 to ASM8 Source Conversion Utility (CONASM)
9. Pertec to or from MicroDisk Transfer Utility (PERTEC)
10. Cassette to or from MicroDisk Transfer Utility (TAPED)
11. Memory Test Utility (MEMTST)
12. Diskette Format Utility (FRMT)
13. Instructions for MicroDOS (HELP)
14. Twelve User Functions

On ROM (UT71)

1. Disk Loader
2. I/O Transfer Routines (READ, WRITE)
3. UT71 Self-Test Routine

Files and File Names

All user-generated programs stored on diskette are identified by file names of up to nine alphanumeric characters. The names for these files are devised and assigned by the user. Each diskette maintains a dynamic directory of all user files kept up to date automatically by the MicroDOS Operating System. Access to a user file is by its name only; the user has no need to know where a program resides and need not maintain track number information for any of the programs.

The major advantage of the MicroDOS Operating System and its use of file names is that only the Operating System is loaded into memory. All other function files stay on diskette and go into memory only when they are used. This dynamic file management system gives the user maximum service from the MS2000 memory capabilities for programming needs.

A file is composed of a set of sectors grouped into a set of clusters. Each cluster contains one sector. Files are located by MicroDOS only on one disk and are identified by name, extension, and device unit number.

The file name consists of from one to six alphanumeric characters and an extension consisting of from one to three alphanumeric characters. The first character of the file name and the extension must be alphabetic. The standard format for a file name is given by the following example:

FILEN1.SXX:#
where FILEN1 is a 1 to 6 character name
SXX is a 1 to 3 character extension, and
is the number of the drive unit (either 0 or 1)

All the MicroDOS system commands are files on the system diskette. These commands are brought into execution when the command name is typed on the console input. Because the main Operating System

resides in memory in locations 9000-BFFF, its area cannot be used by any program. Care must be taken, therefore, not to write a program that uses that area. The majority of memory, however, is left available for execution of the system commands or the user programs. Once a system command or user program has finished operation, the memory area used is returned to the system so that other programs can use that same area.

All file names are stored on a special area of a diskette. This special area is called the Directory and is not the same as the DIR.CM utility which is discussed later in this manual. The Directory resides on track 0 of all diskettes and cannot be deleted. Any diskette that is to be used by MicroDOS must have this file. It can be generated only by the SYSGEN command. Thus, each new diskette must be initialized using the SYSGEN command before it can be used.

MicroDOS supports two types of files: ASCII and binary. ASCII files contain only ASCII characters. Examples are assembly source and object files. Binary files contain only binary information and are used for system programs such as the Assembler and Editor. Binary files require only half the space for storage and can be loaded twice as fast as their ASCII equivalents. Files generated by the system, however, are ASCII unless they have been created by use of the program CDSBIN, which converts an ASCII object file to binary.

A file called the Operating System appears in the Directory as OP.SYS and is designated as file type 3. This file is the actual MicroDOS Operating System and cannot be copied or merged. It can be deleted if the delete protection is removed with the RENAME command. It resides on tracks 1 through 3 and is also transferred only by the SYSGEN command. The information in this file is in binary. The Operating System does not have to be on a MicroDOS diskette. It only has to be on the diskette that is used to load MicroDOS. Not having the Operating System on the diskette frees three tracks for user information, approximately 4% of the diskette area. By means of the DIR command with S option, the presence of the Operating System on a diskette can be ascertained.

Diskettes and Diskette Handling

The diskettes used by MicroDOS are of the double-density type and can store over 322,000 bytes. The drive mechanism has two drive units (the left hand one is designated 0; the right hand one is designated 1). The system has a capacity of over 644,000 bytes of on-line storage.

To assure trouble-free reading and writing files, the diskettes, although fairly rugged, must be handled and stored with care. To avoid damage to the recording surface and to prevent diskette deformation, the following specific precautions should be carefully observed.

* Close the disk guard cover when not in use.
* Do not touch its recording surface.
* Do not smoke when handling the diskette.
* Do not clean the recording surface.
* Do not bend the diskette or deform it with paper clips or other similiar mechanical devices.

The operating and storage environment must be compatible with the materials of the diskette. The environment of the diskette should meet the following criteria:

* No noticeable dirt, dust, or chemical fumes in the immediate area.
* Temperature between 50° F (10° C) and 115° F (45° C).
* Relative humidity between 8 and 80 percent.
* Maximum wet-bulb temperature of 85° F (30° C).
* No direct sunlight on diskette surface for prolonged periods.
* No nearby magnetic fields.

Loading a diskette into a drive mechanism and removing it requires a few precautions to avoid damage and to assure proper operation. These precautions include:

* Do not insert or remove a diskette unless power is applied to the System.
* Insert diskette with read/write access slot first.
* Insert diskette until it automatically becomes locked in.
* Do not remove a diskette from a drive if the select light for that drive shows any sign of activity.
* Format each new diskette with the FRMT utility and then initialize it with the SYSGEN utility
* Do not leave diskette idling in system for prolonged periods.

Memory Requirements

MicroDOS requires memory in the following areas:

	Hexa-Decimal Address	Decimal Address
User Areas	0000-7FFF	0-32767
	C000-FFFF	49152-65535
Utility Program	8000-8FFF	32768-36863
Operating System Area	9000-BFFF	36864-49151

The user area (0000-7FFF and C000-FFFF) is used by either the user programs or by MicroDOS commands. The memory area from 9000 to BFFF is reserved for MicroDOS.

Utility Program UT71

The Utility Program UT71 contains the bootstrap program that initially loads the Operating System into memory. It may be loaded from drive 0 with the "L" command or from any drive with the "B" command. If the specified drive does not contain a diskette, an error message is printed and control remains with UT71. To load the Operating System, place the system diskette in drive unit 0 and type L.

After the Operating System has been loaded, control is transferred to it. If the user wishes to use the debug feature in UT71, the user must press the RESET/RUN U key or return to the UT71 by typing U,8000. If the user is operating under UT71 and wishes to return to the Operating System, which was previously loaded, he must type P9000(CR).

Peripheral Devices

All communications between the peripheral devices is handled by either UT71 or the Operating System. Whenever the command interpreter requires I/O, it goes to the appropriate routine in UT71 or MicroDOS where the function takes place. When the function has been completed, control returns to the command interpreter. Usually the user will not have to be concerned with the peripheral devices because communication with them is handled by MicroDOS automatically.

Program Creation and Translation

With the Editor, the user can create or modify an existing program. The program may be stored on the diskette under a file name with or without an extension. Once the source file has been created on the diskette, it can be input to the Assembler or Editor by referring to its file name.

To speed the loading of object file modules and save space on the diskette, MicroDOS has a command that converts ASCII-HEX object files into binary object files (CDSBIN).

How MicroDOS Operates

Resource Management

A major function of MicroDOS is to manage the resources of the development system so that the user does not have to. MicroDOS provides these functions by having a fixed way of identifying each file on the diskette and the peripheral devices such as the console or line printer.

Device Name Format. With MicroDOS, a specific name is assigned to each peripheral generic device. The device name always begins with the symbol "#" and includes two additional characters. The generic device names pre-assigned by MicroDOS include:

#TY Teletypewriter console printer
#KB Console keyboard
#LP Line printer
#SC CRT screen

Additional names for other peripheral devices can be assigned by the user. A device name for the disk drive mechanism is not needed because its designation is implicit in the file name format.

File Name Format. Each file to be stored on the diskette is identified by a three-part designation consisting of a NAME, an EXTENSION, and a DRIVE NUMBER. Fig. 2 shows the format for assigning identifying designations to files. In this format, NAME is a user-assigned name consisting of an alphabetic character followed by up to five alphabetic or numeric characters.

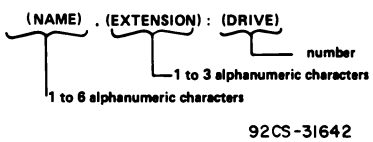


Fig. 2 - Format for naming files.

The EXTENSION, separated from the NAME by a period, may be used to differentiate versions or revisions of the same program. The EXTENSION is one to three alphanumeric characters the first of which, like the NAME, must be alphabetic. Although an EXTENSION is not required when a file designation is assigned, if an EXTENSION is added it must be used every time the file is referenced. When the command CDSBIN is used, if an EXTENSION is not specified by the user, MicroDOS will assign one (CM).

The DRIVE portion of the file designation is a number, either 0 or 1, preceded by a colon(:) and is the logical number of the drive unit. If the DRIVE number is not specified, MicroDOS assumes it is 0 except for the Editor and Assembler. If the file does not reside in the unit specified, an error message is printed.

Whenever FILENAME is used throughout this manual, it means:

::=<NAME>[.<EXTENSION>] [:<DRIVE>]

Examples of FILENAMES are:

AB
AB.XY
AB.XY:0

"Wild-Card" Construct. When a directory is being searched for a file name, the user can take advantage of the "wild-card" construct with certain commands to broaden the search. The "wild-card" construct refers to the use of an asterisk * in the place of some or all the characters in a name or extension. The asterisk means match anything when the directory is being searched. For example:

NAME.* - means match any file name with NAME and extension or without an extension.
*.EXT - means match any file name with EXT and any name.
. - means match any file name.

The asterisk can also specify a wild-card match for the remainder of the name or extension. For example:

AB*.HEX - means match any file name with AB as first two characters of the name and HEX as the extension. These file names would match:

ABC.HEX, ABXYZ.HEX, AB.HEX.

Referencing Files. The MicroDOS method of referring to files by means of a user-selected name that can be both brief and mnemonic can save the user a great deal of time as compared to a physical retrieval and defining of the unit number and track number for a file. MicroDOS keeps track of where the file was established and where it is located on the disk. The file name is converted by MicroDOS to physical addresses for the system to use when the file is opened.

The opening of a file reserves a table for referencing the file and for holding pointers to the file's beginning. As the user accesses the file, the pointers change. The system or the user program may continue to reference this file until it is closed. When one of the system commands (such as VERIFY) makes access to files, the opening and closing of files are done within the command. If the user writes a program that opens or closes files, the program must contain the open and close function. Refer to the chapter entitled **MicroDOS User Functions**, for more details.

Development Station Console. The console is used to echo the user input, display messages that direct the user to perform specific functions, or display data. It may be either a hard-copy terminal or a CRT terminal and is used to communicate with MicroDOS. The designation for the console input device is #KB and is actually the console keyboard. The designation for the console output device may be either #TY for a hard-copy terminal or #SC for a CRT terminal.

When #SC is selected as the output device and when a large data file is sent to the CRT screen, only 22 lines of data will be displayed at a time. The prompt "****" will also appear at the bottom of the screen indicating that more data is to follow. The user may view the next 22 lines by pressing the space bar. This procedure is repeated until the entire file or message has been viewed.

A program that can be halted with the BREAK key (EXAM, COPY, etc.) can usually be either aborted with the Q key or continued with any other key after it has been halted by the BREAK key.

Command Interpreter

The command interpreter is the main interface between the user and the Disk Operating System. The user enters commands through the main console device. Prior to command entry, however, the Operating System has to be loaded into memory from disk. The Operating System is designated MicroDOS VV.RR, where VV is the version number and RR is the revision number. MicroDOS tells the user that it is ready for more input, after it is loaded, by the single prompt ">". At this point, interrupts are disabled. If the user's program sets interrupts and returns to MicroDOS through the system function CDENT, interrupts remain as set by the user's program. If the user reenters MicroDOS through P9000, interrupts will be disabled. Once MicroDOS is executed either by loading with the L command or by executing a P9000 from UT71, interrupts are disabled. Entering MicroDOS any other way will leave the interrupt state as the user program assigned them.

The command to the Operating System includes the name of the system file to be executed plus any parameters or options that the file may need. Because all commands are names of files stored on the disk, the user may add to the existing set of commands very easily.

Command Format. The format for the command line is given by:

<FILENAME>[<DELIM>
<IDENTIFIER>]
[;<OPTIONS>]

where

<FILENAME> is of the form defined in Fig. 2
<DELIM> is a non-numeric character such as comma, space, or slash
<IDENTIFIER> is either another file name or a generic device name

<OPTIONS> are either one or more <IDENTIFIER> or a <NUMBER> depending on the command

All system commands are given the extension "CM". If the user does not type an extension with the filename when specifying a command, MicroDOS will assume that it is "CM". A command cannot have a blank extension. When the user wishes to load an object file with a blank extension, he must add an extension after the file name with the RENAME command. The unit number default value is 0, unless otherwise specified.

When a file is loaded, one of three actions is taken. (l) If the file is a binary file created by CDSBIN, the file is loaded and executed at the starting address given by the CDSBIN program. (2) If the file is an ASCII-HEX file, with no $U information at the end of the file, the file is loaded and control is passed to the command interpreter. To execute the loaded file, the user must press RESET/RUN U followed by a P and execution address command on the console device. (3) If the file is an ASCII-HEX file with the $U information at the end of the file, such as a listing or hex file created by the Assembler, the file will be loaded and executed at the address following the $U.

<DELIM> between file names in the command must be non-alphanumeric characters (such as Δ or = or / or ,) that are not used by the file name.¢ The following commands, therefore, would all perform the same function.

```
DIR MEM.SOH
DIR=MEM.SOH
DIR/MEM.SOH
DIR,MEM.SOH
```

In addition to the above delimiters, MicroDOS ignores leading spaces of a command and treats multiple spaces between commands as one delimiter.

If the file name is not found on the system, the

"FILENAME NOT FOUND"

message will be printed. If an erroneous file name such as ?.# is typed, the message

WHAT?

is typed and control is returned to the command interpreter. The CTRL-C character (03) will cause deletion of the entire command line. The LF character (0A) will type the current contents of the command line.

The rubout key (7F) will print a left bracket "[" followed by the deleted character. When the key for non-delete character is pressed, a right bracket "]" is printed followed by the pressed character. The rubout deletes the last character entered into the buffer. NOTE: Unless otherwise specified, all console inputs are terminated by a carriage return (CR). Note also that corrections cannot be made by backing the cursor and typing over the erroneous characters.

To pass control from MicroDOS to the Utility Program UT71:

Type U,8000 (CR) or press the RESET/RUN U key.

To pass control from UT71 to MicroDOS:

Type P9000 (CR)

Error Messages. All error messages are displayed in a text manner. If a file name cannot be found, MicroDOS prints a message giving the file name requested and stating that it was not found. Recovery from error message depends on the MicroDOS program being executed. Subsequent chapters of this Manual explain the recovery from certain error messages and provide a listing of the error messages along with a description. The description aids in leading the user to a recovery procedure. A list of the MicroDOS error messages is given in Appendix D.

Diskette File Management

File Types. All data on the disk are in a combination of ones and zeroes. In different files, however, the combination of one and zero bits is interpreted in different ways. The Assembler and Editor, for example, create ASCII files and accept only ASCII files. The use of other types of files, such as binary, would yield unpredictable results. ASCII files may be printed. Other files on disk may have some printing result but they will probably be unreadable. For loading purposes, ASCII-HEX files must have an address associated with the object code.

Transferring a file from ASCII-HEX to binary is performed by the CDSBIN program. The resultant binary files consist of only a machine language representation of the program. There are no addresses in the file because all address information is in the file's descriptor area. An ASCII-HEX file, therefore, cannot be loaded as a binary file.

Some of the programs in MicroDOS such as CDSBIN add specific extensions to the file. Its default extension is CM. The other programs, however, such as the Editor, do not have any default extensions. Their default extension is three blank characters.

File Attributes. The attributes that may at the user's option be associated with a disk file include:

1. System (invisible)
2. Write protection

¢ Symbol Δ is used here to indicate a blank space.

3. Delete protection
4. Contiguous

When a file is created, all attributes are usually false or not set. By means of the RENAME command, all the attributes except contiguous may be set or reset. Contiguous must be set when the file is created.

A **system** file is one that is constantly used, such as the Assembler or Editor. These files do not appear in Directory lists and are not members of deleted sets unless a special option is selected when the DIR or DEL command is used.

Write protection is set so that a file cannot be written to. This protection prevents the user from inadvertently destroying a file.

Delete protection is set so that a file cannot be deleted with the delete command. To delete a delete-protected file, the user must first unprotect the file with the RENAME command and then delete the file.

A **contiguous** file is one that is stored without interruption in a set of contiguous sectors. The only file in the system that must be contiguous is the binary file because of the manner in which binary files are loaded by the operating system.

Diskette Structure

Refer to Appendix A for details on diskette organization and structure.

MicroDOS Commands

Files on a disk can be manipulated by the user with either the system functions or the system commands. This section deals only with file manipulation by means of the system commands. The system functions are discussed later.

MicroDOS commands perform the following operations:

1. Format new diskette
2. Initialize new diskette
3. Load and execute programs
4. Create, delete, and list diskette files and directories
5. Change file formats

All diskettes that have never been used are completely blank and must first be formatted with the FRMT utility. Once formatted, the diskettes must be initialized with the SYSGEN utility. Complete system diskettes may be generated.

Program loading and execution are performed by entering the file name. If the ASCII-HEX program is not terminated by $UXXXX, control returns to the command interpreter. Control can then be passed to the program by means of UT71 or the MicroDOS U command. If the file is binary, execution starts at the address established by CDSBIN when the file was generated.

Program creation, deletion, and the control operations such as the listing of diskette files and directories are performed by the following commands.

COPY	Transfers data
DEL	Deletes unprotected files
DIR	Displays directory and associated information
EXAM	Displays or modifies actual information on a diskette
FREE	Lists unused areas of the diskette
MERGE	Merges two or more files into one file
PRINT	Transfers data to line printer with more flexibility than COPY command
RENAME	Changes file names and attributes
U	Starts programs from MicroDOS
VERIFY	Verifies one file against another

The use of the Assembler (ASM8) and Editor (EDIT) in the creation of files and the use of additional programs for diskette control and problem diagnosis are covered in later sections.

The changing of file formats and the editing and assembly of files are performed by the following commands.

CDSBIN	Converts MicroDOS ASCII-HEX files to MicroDOS binary files
EDIT	Creates and changes ASCII files
ASM8	Converts source programs in assembly language into executable (hexadecimal) machine code.
CONASM	Converts ASM4 source files into ASM8 source files.
PERTEC	Transfers files from Pertec drives to MicroDisk drives and from MicroDisk drives to Pertec drives.
TAPED	Transfers files from cassettes to MicroDisk drives and from MicroDisk drives to cassettes.

NOTE: Diskette Recovery

If the directory on a system diskette becomes unusable, there is no way of recovering the data on that diskette. The user, therefore, should always keep backup copies of key files.

3. MicroDOS Command Descriptions

This chapter describes in detail each system command available on MicroDOS. The commands included are:CDSBIN, CONASM, COPY, DEL, DIAG, DIR, EXAM, FRMT, FREE, HELP, MEM, MEMTST, MERGE, PERTEC, PRINT, RENAME, SUBMIT SYSGEN, TAPED, U, and VERIFY. ASM8 and EDIT, which are the Assembler and Editor, respectively, are discussed in greater detail in subsequent chapters.

For ease of use, the system command descriptions are given in a standard format which includes the command name, its purpose, its format, its action, error messages, and examples. In the description for each command, the angular braces < and > indicate required inputs. The square brackets [and] indicate optional inputs. The symbol ::= means "is defined to be." In the examples, the underlined material represents printout generated by the system such as prompts ≥ or queries to the user. (CR) means carriage return.

Note: The system diskette is assumed to be in drive 0 in most of the following examples, so that the command name does not have to be followed by a specific drive number. If the system diskette was in drive 1, the command would have a ":1" appended to it.

A listing of all the MicroDOS error messages is given in Appendix D.

1. Command: **CDSBIN**

2. Purpose:

CDSBIN converts an assembler object file, an assembler listing file, or the ASCII-HEX file generated by the memory save program (MEM) file into a binary object file.

3. Format:

CDSBIN<DELIM><CDSFILE>[,<BFILE >][;<OPTION>] (CR)

Both <BFILE> and <CDSFILE> have the form <NAME>[.< EXTENSION>][:<DRIVE>]

where <CDSFILE> is an ASCII-HEX loadable file and <BFILE> will become a binary object loadable file. If <EXTENSION> for <CDSFILE> is omitted, then a blank is assumed. If <BFILE> is omitted, then the name portion of BFILE will be the name portion of CDSFILE and the extension will be CM.

<DRIVE> is assumed to be 0. <OPTION> is used to specify starting address in hexadecimal. <OPTION> default is address 0. Any CDSBIN-generated file will automatically start after it has been loaded. To prevent automatic starting, the user should make the starting address 9005 to return to MicroDOS or 8029 to return to UT71.

4. Action:

The file <CDSFILE> is read to see how much contiguous disk space must be made available. Once the amount is determined, <BFILE> is allocated the required disk space.

The CDSBIN program is located in memory from F9A3-FFFF. If the user wishes to create a binary file that resides in this area, he must change the origin statement (starting point) ORG in the CDSBIN source file (CDSBIN.SR), reassemble the program, and create a new binary file using CDSBIN with the correct starting address specified in the CDSBIN command. With this new version of the CDSBIN program, the user may create the desired binary file.

If the starting address is specified, the specified address will override the address in the $U record. The address must be a valid hexadecimal number in the range 0000-FFFF, and it must be contained in the memory region spanned by BFILE. If not, an error message is printed.

5. Error Messages:

<FILENAME> F.N NOT FOUND	<CDSFILE> not found
<BFILE> DUP F.N.	
COMMAND SYNTAX ERR	
DISK FULL	No room is available for <BFILE>
FORMAT ERROR	<CDSFILE> did not have the correct format
LOG EOF	A DC3 was not part of the <CDSFILE>
INVALID FILE TYPE	<CDSFILE> was not of file type ASCII

6. Examples:

From an ASCII-HEX file AHFILE located on unit

0, generate a binary file on unit 0 called AHFILE.CM. The execution address is to be 0.

```
≥ CDSBIN,AHFILE(CR)
≥
```

From the same ASCII-HEX file, generate a binary file on unit 1 with the name AH.XY and the execution address of 1000 (hex).

```
≥ CDSBIN,AHFILE,AH.XY:1;1000(CR)
≥
```

1. Command: **CONASM**

2. Purpose:

CONASM allows a program written in ASM4 source code to be used with the ASM8 assembler. The ASM4 source code must be error free.

3. Format:

CONASM<DELIM><FILENAME1>,<FILENAME2>[;<OPTIONS>]

<FILENAME1> is the ASM4 input source code file

<FILENAME2> is the ASM8 output source code file

<OPTIONS>;N Warnings and errors will not be inserted into output as comments

4. Action:

In a single pass, the syntax of the ASM4 source file is modified to conform to that of ASM8. Where context determines appropriate action, a warning is generated and the most likely use is considered. Where SETC, L'*' (the length operator) are encountered, an error message is generated. Invalid characters also generate warnings and are replaced with the Δ character. System limitations generate fatal errors. Warnings and errors are sent to the console screen (#SC) and, unless suppressed, are inserted into the output as comments.

5. Error Messages:

The following message is generated if the input file name is incorrect.

INVALID INPUT FILE NAME
RETYPE>

The following message is generated if the output file name is incorrect.

INVALID OUTPUT FILE NAME
RETYPE>

The following messages are generated during warning conditions.

** WARNING ** THE NUMBER OF WARNINGS HAS EXCEEDED 65,535
** WARNING ** THE NUMBER OF ERRORS HAS EXCEEDED 255
** WARNING ** AN ORG STATEMENT HAS BEEN CHANGED TO A DS STATEMENT
** WARNING ** AN INVALID CHARACTER HAS BEEN REPLACED WITH Δ
** WARNING ** A LABEL - <LABEL> HAS BEEN TRUNCATED TO 9 CHARACTERS
** WARNING ** THIS MAY NEED TO PRECEDE THE FIRST USE OF EXPRESSION
** WARNING ** A LABEL - <LABEL> HAS BEEN DUPLICATED BY TRUNCATION
** WARNING ** <LABEL> IS A SYSTEM, SOURCE LABEL DUPLICATE

The following messages are generated during error conditions.

*** ERROR *** INPUT LINE EXCEEDS 80 CHARACTERS
*** ERROR *** THE ERROR LIST HAS OVERFLOWED
*** ERROR *** THE OUTPUT LINE BUFFER HAS OVERFLOWED
*** ERROR *** AN UNBREAKABLE LINE TOO LONG HAS BEEN ENCOUNTERED
*** ERROR *** UNBALANCED PARENTHESES
*** ERROR *** NO LABEL FOUND WHERE EXPECTED
*** ERROR *** THE NEW ASSEMBLER CANNOT PROCESS SETC OR L STATEMENTS
*** ERROR *** MISSING QUOTE IN A NUMBER
*** ERROR *** SYMBOL_ TABLE OVERFLOW

The following messages are included at the end of the conversion to show the total number of warnings and errors.

* THERE WERE XXXXX WARNINGS IN THIS CONVERSION
* THERE WERE XXX ERRORS IN THIS CONVERSION

1. Command: **COPY**

2. Purpose:

COPY is a generalized copy routine that can take a data file from one peripheral device to another. It can copy from disk to disk, disk to teletypewriter printer, disk to screen, keyboard to disk, and disk to line printer. It can copy either ASCII or binary.

3. Format:

COPY<DELIM><NAME1><DELIM><NAME2> (CR)

<DELIM> is a command line delimiter

<NAME1> is the name of the source file or source

device, and <NAME2> is the name of the destination file or destination device.

If <NAME1> is a disk file name, it is of the format <NAME1>[.<EXTENSION1>][:<DRIVE1>] and <NAME2> must be specified.
If <DRIVE1> is not specified, "0" will be used.
If <EXTENSION1> or <EXTENSION2> is not specified, blank will be used.
If <NAME2> is a disk file name, it is of the format <NAME2>[,<EXTENSION2>][:<DRIVE2>]

The following are mnemonics for the non-disk devices used with the command COPY:

#LP Line printer
#TY Teletypewriter printer
#SC Console screen
#KB Console keyboard

4. Action:

Three types of file copying can be requested:
Disk to disk
Disk to device
Device to disk

Disk-to-disk copy takes the information associated with one file name and copies it to the other file name. Both file names must be specified. Disk- to-device copy is a transfer from a disk file to a line printer or console printer. This transfer permits the printing of a disk file. Device-to-disk copy is a transfer from a keyboard to a disk file. Transfer from keyboard to disk file is terminated by entering CTRL-S (EOF).

To pause the transfer of the COPY program, press the BREAK key on the keyboard. To abort COPY after a pause, press the Q (QUIT) key. Any other key will continue the copying. Note: When #TY or #SC are used, both will output the file to the CRT screen. #TY will copy the file onto the screen until the BREAK key is pressed; #SC will only output 22 lines and then stop. To continue, the space bar must be pressed.

5. Error Messages:

<FILENAME>	F.N. NOT FOUND <NAME1> does not exist.
DIR FULL	No more room exists for another file name in the directory.
DISK FULL	No more room exists for file on disk. Some of the data may have been transferred.
INVALID FILE TYPE	Disk file being copied to a non-disk device has a file type other than ASCII or ASCII-HEX format. COPY cannot dump non-ASCII files to an ASCII device. The Operating System or any operating system file cannot be copied.
INVALID DV	Disk was entered (e.g., #DK).
NO SUCH DV	Peripheral device specified does not exist in system.
INVALID DATA TRANSFER TYPE	Device requested does not transfer data in the direction requested (e.g., copy to an input-only device or copy from an output-only device).
COMMAND SYNTAX ERR	A name contained a wild card construct, or no file name was found as the first or second parammeter.

6. Examples:

Copy the ASCII file ASCII to the screen.
≥ COPY,ASCII,#SC (CR)

Copy the ASCII file ASCII on unit 0 to the ASCII file ASCII on unit 1.
≥ COPY,ASCII,ASCII:1 (CR)

1. Command: DEL

2. Purpose:

DEL deletes MicroDOS file names from a directory and de-allocates all disk space belonging to the deleted file. A single file, a list of files, or a family of files may be deleted.

3. Format:

DEL<DELIM><NAME>[.<EXTENSION>][:<DRIVE>][;<OPTION>](CR)
<DELIM> is a command line delimiter;
<NAME> specifies a file name, a list of file names, or a family of file names,
<EXTENSION> specifies an extension or a family of extensions, and
<DRIVE> is the logical drive number.
<OPTION>;S includes files with the system attribute when deleting.

If the S option is not chosen, system files will not be deleted and
<FILENAME> F.N. NOT FOUND
error message will be displayed.

4. Action:

The list of file names specified on the command line are searched for in the specified directories. If a specified file is not found in a directory, the message

<FILENAME> F.N. NOT FOUND

will be displayed. Otherwise, the message

<FILENAME> DELETED

will be displayed.

If a file to be deleted has the delete-protection attribute set, the message.

<FILENAME> IS DELETE PROTECTED

will be displayed. Protected files cannot be deleted until their protection has been removed (see RENAME command). Control will be passed back to the operating system when the last file name has been deleted.

5.Error Messages:
See 4. Action, above.

6. Examples:
Delete file XYZ on unit 1 and QST on unit 0.
≥ DEL,XYZ:1,QST:0(CR)

Delete all files having the extension A1 on unit 0, file A1 on unit 1, and all files with the first two letters XY on unit 0.
≥ DEL,*.A1,:1,XY*.*(CR)
Delete system file ABC.
≥ DEL,ABC;S(CR)

For an explanation of the * symbol, see "Wild-Card" Construct in the chapter entitled **Understanding MicroDOS.**

1. Command: **DIAG**

2. Purpose:
DIAG is a diskette diagnostic program that provides the facility of detecting media errors. These errors are called CRC (Cyclic Redundancy Check) errors. They indicate that the Read and Read CRC operations result in the detection of a possible data error. Some CRC errors can render a diskette unreadable by the Editor and Assembler. DIAG provides the user the option of attempting to fix such errors.

3. Format:
DIAG(CR)

4. Action:
Each sector on the diskette contains CRC bytes. These CRC bytes are generated from a cyclic permutation of data bits starting with bit 0 of the first byte and ending with bit 7 of the last byte. When data is read from a diskette, status bits are checked by the diagnostic program. The Floppy Disk System hardware automatically computes the CRC during a read operation, and if an error is found, the CRC status bit is set at the end of the read.

The diskette diagnostic program DIAG seeks each sector (starting from sector #01 of track #00 of the selected drive), and the Read CRC command is issued. If a CRC error is detected, the CRC Read operation is repeated. There are two types of CRC errors that can be detected:

1. A "SOFT" ERROR is an error that can be recovered by data rereading. A marginal diskette is indicated if many soft errors are present.
2. A "HARD" ERROR is an error that can not be fixed even by rereading.

If a CRC error is detected, 16 attempts to reread the data are made. If a successful read is made, the error will be labeled as being "soft". If the 16th attempt also fails, the error is considered "hard". After any detected error, the program prints a message giving the track number, sector number, and the type of the CRC error detected. A fix-up option is also provided to attempt fixing hard errors by rewriting the data back into the sector. If a hard error is not fixed by data rewriting, the user receives an "error-not-fixed" message and the specified sector should not be used for data storage. If the CRC error is corrected by data rewriting, the "error-fixed" message is printed and the specified sector can be used for data storage.

It should be noted, however, that a sector so "fixed" may now contain data not exactly the same as that which was originally intended. Because a CRC error was detected, some data was recorded incorrectly. Data rewritten by the fix-up routine attempts to remove the CRC discrepancy, but cannot correct a garbled byte. Thus, a file so fixed should be visually inspected for corrections and fixed by means of the EXAM program.

In the case of either a hard or soft error, the program continues processing the rest of the sector on the diskette.

Any diskette exhibiting errors has become marginal and should be copied immediately and the marginal disk discarded. The user can abort the program while it is testing the diskette by pressing and holding the BREAK key.

5. Error Messages:
In addition to the errors described under 4 above, if the drive fail bit is set, the "CK DRIVE" message is issued. In this case, the program has to be restarted.

6. Example:
≥ DIAG(CR)
DISKETTE DIAGNOSTIC PROGRAM
FIX UP ? Y(CR)
ENTER UNIT NUMBER: 1(CR)
UNIT: 1
TRACK: 00 SECTOR: 01 SOFT ERROR
TRACK: 35 SECTOR: 08 ERROR FIXED
TRACK: 69 SECTOR: 09 ERROR NOT FIXED
TEST DONE
≥

1. Command: **DIR**

2. Purpose:
DIR displays MicroDOS file names from a directory. An entire directory or selective parts of it may be displayed on the console screen. The minimum directory information displayed is a file name and extension. At

the user's option, an entire directory entry, in addition to its allocation information, can be displayed.

3. Format:

DIR<DELIM>[<NAME>[.<EXTENSION>][:<DRIVE>]][;<OPTIONS>](CR)

<DELIM> is a command line delimiter,

<NAME> specifies a file name or a family of file names,

<EXTENSION> specifies the extension or a portion of the extension,

<DRIVE> specifies the logical disk drive number and

<OPTIONS> specifies one of the following defined actions:

E - Displays entire directory entry information (attributes, starting sector number, file size, and directory entry number).

A - Displays complete allocation description for each file name (segment descriptors).

L - Displays directory information on the line printer.

S - System files (files with 'S' attribute) may be included when a family of files is displayed.

4. Action:

The disk directory specified by <DRIVE> is searched for the specified <NAME> and <EXTENSION>. If the drive number is omitted, drive 0 will be selected. If only the drive number is specified (explicitly or implicitly), all directory entries other than system files on that drive will be searched for. Directory entries found by the above search procedure will be displayed on the system console unless option 'L' is specified. The following format will be used to display the directory entries:

DRIVE: DISKID:
NAME.EXTENSION[<ATTR><SSN>
<SIZE><DEN>]

<ATTR> is a list of attributes,

<SSN> is the number of data sectors actually used, and

<DEN> is the entry's directory entry number (index to physical location in directory). <DEN> consists of two hexadecimal digits (an 8-bit binary number). The upper four bits are the physical sector number within the directory. The lower four bits are the entry's physical position within a directory sector (0-7). These quantities are displayed only if the 'E' or 'A' option has been specified.

<DISKID> is taken from the special ID sector. See SYSGEN command for information on DISKID. Refer to Appendix A for details of diskette organization.

<ATTR> is always displayed as a six-character field of the form:

WDSC.# Each position contains either a letter or a period '.' indicating the presence or absence of that attribute, respectively. The following meanings are associated with the specific attribute positions:

W = write protection
D = delete protection
S = system file
C = contiguous allocation
# = file format - a digit from 1 to 3
1 = > binary
2 = > ASCII
3 = > Operating System

After all directory entries from the search have been displayed, the message

TOTAL NUMBER OF SECTORS: YYYYY
TOTAL DIRECTORY ENTRIES SHOWN: XXXXX

will be displayed. XXXXX is a decimal count of the displayed directory entries. YYYYY is the sum of the size of all displayed files (decimal sectors). YYYYY is displayed only if the E or A option is used. If no directory entries are found in the search, the message

NO DIRECTORY ENTRIES FOUND

will be displayed. After all entries returned by the search are displayed, control will be returned to the command interpreter.

If the A option is specified, the information contained in a file's first sector (sector pointer block) will be displayed in addition to the full directory entry. Following each displayed directory entry will be one line of allocation information for each segment of the file. The format follows:

SEG I SECT J SIZE K

where I is the segment number, J is the sector number that starts the segment, and K is the number of allocated sectors in the segment.

5. Error Messages:

See 4 above.

6. Examples:

Get a listing of the directory on unit 0.
≥ DIR (CR)

See if file QRS is on unit 1.
≥ DIR,QRS:1(CR)

List the directory information of all files on unit 0

with the extension CM.

≥ DIR,*.CM;E,S(CR)

List on the line printer all the allocation information for all files on unit 0.

≥ DIR,*.*;A,L(CR)

For an explanation of the * symbol, see "Wild-Card" Construct in the Chapter on **Understanding MicroDOS.**

1. Command: **EXAM**

2. Purpose:

EXAM is a utility program that allows examination or modification of information on a diskette.

3. Format:

EXAM<DELIM>[;<OPTION>](CR)

<DELI M> is a command line delimiter, and

<OPTION> is L if the header and data are to be printed on the line printer.

4. Action:

After printing a header, the program asks for various parameters such as drive, track, sector, filename, physical sector, or logical sector depending on which mode is selected.

EXAM can operate in one of three modes. In the UNIT/TRACK mode, the user enters the drive, track, and sector that he wishes to examine or modify. In the PHYSICAL mode, the user enters the drive and the physical sector number. In the LOGICAL mode, the user enters the drive, the filename, and the logical sector number.

Each 512-byte sector is displayed as two 256-byte screens. The top of each screen displays a header containing decimal values that show such information as drive, track, sector, physical sector, filename, or logical sector depending on which mode is selected. The left side of each screen shows the position of each byte within the sector. The right side of each screen shows the ASCII equivalent of the data bytes. All non-printing data bytes are presented as a '.' in this area. The bottom of each screen displays a menu of possible operations that can be performed after viewing a screen.

The user can halt the program while it is displaying data by pressing and holding the break key. After the program halts, it can be resumed by pressing the space bar. If the Q key is pressed, the program will revert to the menu.

If the modify function is selected, the program will ask how to modify the sector that is being displayed. The user can enter new information in either ASCII or hexadecimal. After the program prompts for the new data, the user should enter either MH for modify hex or MA for modify ASCII, a space, a hex number specifying the byte position in the displayed sector to start modifying, a space, and finally the new data. In the MA mode, the ASCII characters will be converted to their hexadecimal equivalents before being changed on the diskette.

5. Error Messages

****** BEGINNING OF DISK ******

Message obtained when the user attempts to access a physical sector with a value less than 0.

****** END OF DISK ******

Message obtained when the user attempts to access a physical sector with a value greater than 629.

6. Example:

Examine physical sector 11 on the diskette in drive 1 and change byte FEH in this sector from a 35H to a 37H.

≥ EXAM(CR)

DISKETTE EXAMINATION PROGRAM

```
ENTER (L) LOGICAL (P) PHYSICAL (U) UNIT/TRACK :P(CR)
ENTER DRIVE NUMBER :1(CR)
ENTER PHYSICAL NUMBER :11(CR)
          DRIVE: 1                                        PSN: 0011
BYTE: 0000  0C21  4D0D 0A30  3030  3020  3B20  2020  2020   .!M 0000 ;
BYTE: 0010  2020  2020 2020  2020  2030  3030  3120  0D30   0001 .0
BYTE: 0020  3030  3020 3B20  2020  2020  2020  2020  2020   000 ;
BYTE: 0030  2020  2030 3030  3220  2E2E  5553  4552  2046   0002 USER
BYTE: 0040  554E  4354 494F  4E20  4558  414D  504C  4520   FUNCTION EXAMPLE
BYTE: 0050  2D20  434F 5059  2041  2046  494C  4520  544F   - COPY A FILE TO
BYTE: 0060  2041  4E4F 544B  450D  0A2E  5220  4649  4C45   ANOTHER FILE
BYTE: 0070  2E0D  3030 3030  203B  2020  2020  2020  2020   0000 ;
BYTE: 0080  2020  2020 2020  3030  3033  202E  2E54  4845   0003 THE
BYTE: 0090  2046  4F4C 4C4F  5749  4E47  2049  4E46  4F52   FOLLOWING INFOR-
BYTE: 00A0  4D41  5449 4F4E  2049  5320  4120  4445  4649   MATION IS A DEFI-
BYTE: 00B0  4E49  5449 4F4E  2046  4F52  0D0A  2E20  5448   NITION FOR THE
BYTE: 00C0  4520  5052 4F47  5241  4D3A  0D30  3030  3020   PROGRAM 0000
BYTE: 00D0  3B20  2020 2020  2020  2020  2020  2020  2030   ; 0
BYTE: 00E0  3030  3420 2E2E  0D30  3030  3020  3B20  2020   004 0000 ;
BYTE: 00F0  2020  2020 2020  2020  2020  2030  3030  3520   0005

  (1) AHEAD ONE SCREEN(2) AHEAD ONE SECTOR(3) AHEAD CONTINUOUS
  (4) BACK ONE SCREEN (5) BACK ONE SECTOR (6) BACK CONTINUOUS
  (7) MODIFY (8) NEW PSN (9) RESTART (A) EXIT
ENTER NUMBER OF DESIRED FUNCTION :7(CR)
ENTER NEW DATA :MH FE 37(CR)
          DRIVE: 1                                        PSN: 0011
BYTE: 0000  0C21  4D0D 0A30  3030  3020  3B20  2020  2020   !M 0000 ;
BYTE: 0010  2020  2020 2020  2020  2030  3030  3120  0D30   0001 .0
BYTE: 0020  3030  3020 3B20  2020  2020  2020  2020  2020   000 ;
BYTE: 0030  2020  2030 3030  3220  2E2E  5553  4552  2046   0002 USER
BYTE: 0040  554E  4354 494F  4E20  455B  414D  504C  4520   FUNCTION EXAMPLE
BYTE: 0050  2D20  434F 5059  2041  2046  494C  4520  544F   - COPY A FILE TO
BYTE: 0060  2041  4E4F 5448  450D  0A2E  5220  4649  4C45   ANOTHER FILE
BYTE: 0070  2E0D  3030 3030  203B  2020  2020  2020  2020   0000 ;
BYTE: 0080  2020  2020 2020  3030  3033  202E  2E54  4845   0003 THE
BYTE: 0090  2046  4F4C 4C4F  5749  4E47  2049  4E46  4F52   FOLLOWING INFOR-
BYTE: 00A0  4D41  5449 4F4E  2049  5320  4120  4445  4649   MATION IS A DEFI-
BYTE: 00B0  4E49  5449 4F4E  2046  4F52  0D0A  2E20  5448   NITION FOR THE
BYTE: 00C0  4520  5052 4F47  5241  4D3A  0D30  3030  3020   E PROGRAM : 0000
BYTE: 00D0  3B20  2020 2020  2020  2020  2020  2020  2030   ; 0
BYTE: 00E0  3030  3420 2E2E  0D30  3030  3020  3B20  2020   004 0000 ;
BYTE: 00F0  2020  2020 2020  2020  2020  2030  3030  3720   0007

  (1) AHEAD ONE SCREEN (2) AHEAD ONE SECTOR (3) AHEAD CONTINUOUS
  (4) BACK ONE SCREEN (5) BACK ONE SECTOR (6) BACK CONTINUOUS
  (7) MODIFY (8) NEW PSN (9) RESTART(A) EXIT
ENTER NUMBER OF DESIRED FUNCTION :A(CR)
≥
```

1. Command: **FRMT**

2. Purpose: FRMT is used to format a new diskette or one that that has been damaged by a magnetic field. It will completely erase all previous headers and data, write a new header for each sector, and fill each sector with its corresponding track value in hexadecimal. It verifies each track and reports errors.

All diskettes will be formatted with double-density, 512 bytes per sector and nine sectors per track (numbered 1 to 9). The user may specify drive number, (0-3, defaults to 1), number of tracks (70 or 80, defaults to 70), and single or double-sided (defaults to single).

3. Format:
FRMT(CR)

4. Action:
FRMT prints the following message:

RCA MICRODISK FORMAT PROGRAM
DEFAULT VALUES:
DRIVE # = 1, # OF TRACKS = 70, # OF SIDES = 1
FORMAT, CHANGE/PRINT DEFAULTS, OR QUIT (F, C, P, OR Q)?

If the user presses the C key, the program will prompt for new drive number, number of tracks, and single- or double-sided. Keys outside the specified ranges will be ignored except for (CR) which will return to the menu.

If the user presses the P Key, the present set of default values for drive, number of tracks, and single- or double-sided will be printed.

If the user presses the Q key, the program will return to MicroDOS.

If the user presses the F key, the program will prompt with:

OK TO FORMAT DRIVE X (Y/N)?

(X is the selected drive number). If the user responds with any key but Y, the program will return to the menu. If Y is pushed, the diskette in the selected drive will be formatted and verified. If a drive not ready or write-protected condition is found, the program returns to the menu. If a track does not successfully format and verify on the first try, but does within 5 tries, a soft error message and the track number will be printed. If the track cannot be verified in 5 tries, a hard error message is printed. Pushing the break key at any time will abort the operation.

5. Error Messages:
DRIVE NOT READY DURING (ACTION), TRACK XX
A Drive-not-ready signal was encountered. The (ACTION) could be a SEEK, FRMT, or VERIFY attempt. The track number is in decimal.

DRIVE OR CONTROLLER FAILED DURING (ACTION), TRACK XX.
A drive fail signal was encountered during (ACTION).

DISKETTE WRITE PROTECTED DURING FORMAT, TRACK XX.
A write protect signal was encountered when attempting to format.

SOFT ERROR DURING VERIFY, TRACK XX.
A CRC or other disk read error was encountered during the CRC READ. It was recovered within 5 tries.

HARD ERROR DURING VERIFY, TRACK XX.
An error as above was not correctible within 5 tries.

TERMINATION ERROR DURING (ACTION), TRACK XX.
An otherwise unidentified error was encountered.

1. Command: **FREE**

2. Purpose:
FREE informs the user how many non-allocated sectors remain on the disk and how many unused directory entries are available.

3. Format:
FREE[<DELIM>:<DRIVE>] (CR)
<DELIM>is a command line delimiter, and
<DRIVE>is the logical drive number. If <DRIVE> is not specified, 0 will be assumed.

4. Action:
FREE will cause the following message to be printed on the display:

DRIVE O DISKID:(DATE AND ID FROM ID SECTOR)
XXXXX SECTORS YYYYY FILES

XXXXX and YYYYY are decimal numbers. The maximum number of free sectors on the disk is 620; the maximum number of entries allowed in the directory is 128 if the capacity of the disk will allow this number of files.

5. Error Messages:
None applicable

6. Example:
List on the console the free area of drive 1.
≥ FREE :1(CR)

1. Command: **HELP**

2. Purpose:
HELP is a file that contains instructions for using each of the other MicroDOS commands.

3. Format:
HELP(CR)

4. Action:

After HELP is loaded, a numbered listing of the MicroDOS commands is displayed on the screen. The operator enters the number of the command he plans to use followed by (CR). HELP then displays the instructions for using the selected command.

5. Error Messages:

None applicable

1. Command: **MEM**

2. Purpose:

MEM is used to save on a diskette user object code located anywhere in memory. A memory file thus saved may later be rapidly reloaded into memory. Data is saved in ASCII-HEX format.

3. Format:

MEM(CR)

4. Action:

MEM normally resides in memory FB8C through FFFF. Memory from 0000 to FFFF may be selectively saved by this command. The program is written so that only the first ORG statement need be changed to relocate it. Relocation is accomplished by use of the Editor program to change the ORG statement and the Assembler program to generate object code.

Once assembled, the hex code file should be converted to a binary file by use of CDSBIN. The MEM program is loaded by the command interpreter.

After

MEM(CR)

is keyed in, the program starts by typing the following header message.

DISK SAVE PROGRAM
FIRST ADDR?XXXX(CR)

The user should enter the first address of memory to be saved. The program then asks

LAST ADDR?XXXX(CR)

The user replies to this query with an address XXXX. Memory from the first address up to and including this address is selected for saving.

Next, MEM requests the selection of a disk file name.

WRITE?FILENAME (CR)

5. Error Messages:

See Appendix B, MicroDOS Error Messages.

6. Examples:

Copy a program onto disk that is loaded in memory at location 0000 through 0340. Give it the file name WFIL2. MicroDOS is currently in control.

```
≥ MEM(CR)
DISK SAVE PROGRAM
FIRST ADDR?0000(CR)
LAST ADDR?0340(CR)
WRITE?WFIL2(CR)
≥ CDSBIN WFIL2(CR) ..To convert to binary
                   ..file WFIL2.CM which
                   ..starts execution at
                   ..0000.
```

The saved program can be called from disk at any time by typing its file name WFIL2.CM.

See Appendix C - MS2000 Memory Test. **MEMTST**

1. Command: **MERGE**

2. Purpose:

MERGE copies and merges one or more ASCII files. Its main use is for continued files or multi-diskette files generated by the assembler or the editor. The ASCII files do not have to be terminated by a DC3.

3. Format:

MERGE<DELIM>NAME1>[
<DELIM><NAMEn>]n(CR)
<DELIM> is a command line delimiter,
<NAME1> is the name of the destination file,
<NAMEn> is the names of the files to be merged

The default values for any extension is three blank characters. The default value for any unit number is 0.

4. Action:

MERGE copies the first file from the source list into the destination file name. Whenever a null file name is encountered, the MERGE command adds a DC3 to the output file and closes all opened files. If any of the source file names cannot be found, an error message is printed. MERGE then allows the user to retype the erroneous file name or to exchange disks (i.e., put into the drive being used for the source files the diskette containing the desired files) and retype the file name. MERGE cannot be aborted by pressing the BREAK key. It can be aborted only by typing a (CR) in response to a request to retype file name. The DC3 end-of-file marker will be removed from files before they are merged.

If the destination file name was incorrectly typed or if the file name already exists on the diskette, MERGE will inform the user and allow him to correct the file name or replace the diskette with another diskette. The resulting file name will have the attributes of the first file in the input file list. MERGE should be used only on ASCII or ASCII-HEX files.

5. Error Messages:

SYNTAX ERROR	Either an illegal file name or a wild-card type file name. The user should retype the correct name when prompted. A (CR) will abort MERGE and return control to MicroDOS.
<FILENAME> F.N. NOT FOUND	File name not found on diskette. User should either place diskette containing the file into the drive unit or retype the name specifying that file is in other unit. Only a (CR) is needed to abort MERGE.
DIR. FULL	No more room exists for another file name in the directory.
DISK FULL	No more room exists for file on disk. Some of the data may have been transferred. Delete the incompleted file.
BAD FILE TYPE	A file other than ASCII was called for. The user should type a (CR) to abort merge and return control to MicroDOS.

6. Example:

Merge files SOURCE.X1 and SOURCE.X2 into file DESTFN.

≥ MERGE DESTFN,SOURCE.X1,
SOURCE.X2(CR)

1. Command: **PERTEC**

2. Purpose:

PERTEC is used to copy an ASCII file from a Pertec drive to a MicroDisk drive or from a MicroDisk drive to a Pertec drive. It can also generate a binary file from a hexadecimal or list file in a Pertec drive to a MicroDisk drive.

3. Format:

PERTEC(CR)

4. Action:

When the transfer is from the Pertec drive to the MicroDisk drive, the user should first prepare a source, hexadecimal, or list diskette of the desired input file on an 8-inch diskette. It must be in unit/track format and start on track 0. This requirement can be met by use of the FCOPY command. This diskette must then be placed in Pertec drive 0.

After the program is loaded, it will print the following:

8" TO MICRODISK ASCII OUTPUT (A)
8" TO MICRODISK BINARY OUTPUT (B)
OR MICRODISK TO 8" ASCII OUTPUT?(C)

If the user types an A, the copy-ASCII mode is selected. The program then prompts for the name of the output file that will be generated on the MicroDisk drive. After this name is entered, the entire file is copied from the Pertec drive to the MicroDisk drive.

If the response to the initial prompt had been a B, the generate-binary mode would be selected. The program prompts for the name of the output file and execution address. After these names are entered, the entire hexadecimalor list file is loaded from the Pertec drive into memory. The hexadecimal or list file must have been assembled from a source file that had an END statement. A binary file is then generated on the Microdisk drive. This binary file will automatically start execution at the specified address if it is later loaded from the MicroDisk drive. If no address is specified, execution starts at address 0000H.

If the response had been a C, the program will ask for the input filename on the MicroDisk drive. After the name is entered, it will ask if it is OK to write to unit 0, track 0 on the Pertec drive . If a Y is entered, the ASCII copy will take place.

In any mode, the program can be aborted by pressing the BREAK key. The following will be printed

ABORTED

Control will then be returned to the operating system.

5. Error Messages:

<OUTFILE> DUP F.N.

The specified output file name already exists.

6. Examples:

Copy the ASCII file in Pertec drive 0 to MicroDisk drive 1 and give it the name ABC.XYZ.

≥ PERTEC(CR)
8" TO MICRODISK ASCII OUTPUT (A)
8" TO MICRODISK BINARY OUTPUT (B)
OR MICRODISK TO 8" ASCII OUTPUT
(C)?A(CR)
ENTER OUTPUT FILENAME : ABC.XYZ:1
≥

Convert the list file in Pertec drive 0 to a binary file in MicroDisk drive 0 and give it the name TEST.CM. Specify an execution address of 024CH.

≥ PERTEC(CR)
8" TO MICRODISK ASCII OUTPUT (A)
8" TO MICRODISK BINARY OUTPUT (B)
OR MICRODISK TO 8" ASCII OUTPUT
(C)?B(CR)

ENTER OUTPUT FILENAME AND EXECUTION ADDRESS :TEST.CM;24CH
≥

1. Command: **PRINT**

2. Purpose:
PRINT gives the user a variety of options in outputting one or more files to a line printer.

3. Format:
PRINT<DELIM><FILENAME>[;<OPTIONS>](CR)
<FILENAME>is the name of the file to print. Both drives will be searched if no unit number is specified. If no file name is given, a blank page will be ejected. A file name may actually consist of a list of files to be printed.

<OPTIONS>

H -	Suppress the header that comprises the name of the file and the name of the diskette.
P -	Suppress the page numbers.
Lnn -	Use nn lines per page (0<nn<100)
Snn -	Skip nn-1 lines between each printed line. (1<nn<99)
D -	If a line contains one or more CONTROL-H characters, count it as a double-sized line and adjust the line counter accordingly.
Ttttttt...d -	Print the text, ttttt..., at the top of each page. The delimiter, d, is either an ESCAPE character or the end of the command line.
Cnn-	Print nn copies of each file specified. (0<nn<100)
E -	Exit the print program.
Wnnn -	Width of paper. nnn columns per page. (0<nnn<200). Starts with 117 columns. Remembers last W command.
Xnnn -	Width of paper for this command line, nnn columns per page. (0<nnn<200) Remembers old width and restores it after finishing present command.
N-	Suppress resetting page numbers between files. Action is that page number will continue from lst page of previous file. N command is inoperative if C command is specified.

4. Action:
If both the file name and the options are omitted when the PRINT program is loaded, the program responds with a prompt ":" after which commands can be entered.

5. Error Messages:

INVALID FILENAME	Wild-card format "*" cannot be used
<FILENAME> F.N. NOT FOUND	Specified file not found
01 DR FAIL	Drive number was not specified and drive α1 did not contain a diskette
*** PRINTER NOT READY. CONTINUE OR EXIT (C/E)?	Line printer not ready

6. Examples:
Print a single file and return to MicroDOS
≥ PRINT MEM.SR(CR)
Print a file ABC and suppress the header and page numbers.
≥ PRINT ABC;HP(CR)

With print already loaded, print 5 copies of file report.
:REPORT;C5(CR)

With print already loaded, return to MicroDOS.
:;E(CR)

1. Command: **PROM25**

Operating instructions for this command are given in the technical literature for the PROM Programmer CDP18S680.

1. Command: **RENAME**

2. Purpose:
RENAME allows the names, extensions, and attributes to be changed in a directory. The information in the file remains the same.

3. Format:
RENAME<DELIM><FILENAM1>[,<FILENAM2>][;<ATTR>](CR)
<DELIM> is a command line delimiter;
<FILENAM1> equals <NAM1>[.<EXTENSION1>][:<DRIVE> which is the name of the file for which the name or attributes are to be changed.
<FILENAM2> equals <NAM2>
<EXTENSION2> which is the new file name.
The contents of <ATTR> is the new set of attributes.
If<NAM2> is omitted, then <NAM1> will be used. If the <EXTENSION1> is omitted, a blank will be used. If <DRIVE1> is omitted, 0 will be used.
<ATTR> will be one or more of the following letters having the meanings indicated.

D Set delete protection
W Set write protection
S Set system file program
N Set non-system file program
X Remove delete and write protection

4. Action:

The name or attributes of a file name or a family of file names will be changed. <FILENAM1> must be specified. Either <FILENAM2> or <ATTR> or both must be specified. If <FILENAM2> or <ATTR> is not specified, the message "INPUT FILENAME AND/OR ATTRIBUTES" will be printed requesting the information. If <FILENAM2> is a duplicate file name, a duplicate file name message will be printed and the RENAME command will be aborted. If <FILENAM1> does not exist, a "FILENAME NOT FOUND" message will be printed and the RENAME command will be aborted.

The command line interpreter allows a file name to be specified with the "wild-card" construct "*.*". With the RENAME command, however, only a partial wild-card construct can be used. An asterisk may appear to the left of the period or to the right of the period but it cannot be placed in both positions. If two asterisks are used in this manner, an illegal file name message will be printed and the RENAME command will be aborted. With RENAME, however the complete wild-card construct "*.*" may be used for changing attributes.

If a unit number is associated with <FILENAM2>, it will be ignored and only the unit associated with <FILENAM1> will be used.

5. Error Messages:

See 4. Action, above.

6. Examples:

Change the extension on all file names having the extension DEF to the extension XY.

≥ RENAME*.DEF,.XY(CR)

Change the name of file ABC.XY to XYZ.AB.

≥ RENAME ABC.XY,XYZ.AB(CR)

Make file XYZ on unit 1 delete protected.

≥ RENAME XYZ:1;D(CR)

Remove all protection from files on unit 1 having the extension CM.

≥ RENAME *.CM:1;X(CR)

Change the name of file XYZ to ABC and make it a system file

≥ RENAME XYZ,ABC;S(CR)

Change all file names that have ABC as the name portion of the file name to XYZ, as the new name portion of the file name.

≥ RENAME ABC.* ,XYZ.*(CR)

1. Command: **SUBMIT**

2. Purpose:

SUBMIT is a program that permits sequences of commands to MicroDOS or application programs to be stored in a command definition file and executed. It is especially useful for repetitive operations, and frees the user from keystroke errors and keyboard attendance during serial program execution.

A special command definition file named AUTO.SUB is automatically sought when MicroDOS is initially loaded. This permits the user to define execution of an initial sequence of commands immediately following load of MicroDOS. if AUTO.SUB does not exist, no attempt is made to execute from such a file. Since a search for this file is made on drive 0, the user will notice disk activity on drive 0. Subsequent warm start of MicroDOS from the UT level may bypass execution of AUTO.SUB by starting execution of MicroDOS at address #9005.

A command file language permits additional features during command file execution:

- passes up to 10 parameters at command file invocation time
- types messages to the terminal display (~TYPE)
- directs that input be tken from the terminal keyboard rather than the command file, with resumption of execution from the command file (~LREAD, ~KREAD)
- annotates the command file (~COMMENT)
- exits from the command file to MicroDOS (~EXIT)
- automatically translates dollar sign character ($) to esc character for EDIT
- recognizes the break key to abort command file execution
- detects error calls to CDERR of MicroDOS and recovers by suspending command file execution to give user a choice to either continue or abort
- supports an index (~J) which may take on a range of values from 0 to 99. It may be set, incremented by one, or decremented by one (~SETJ, ~INCJ, ~DECJ)
- controls sequencing through the command file with jumps (~GOTO) and conditional tests (~IF)

3. Format

SUBMIT<delim><filename> [<param> <delim>...](CR)

The <delim> may be a space or comma character. <filename> is the command definition file.

Parameters, up to a maximum of 10, may be passed to the command file when SUBMIT is invoked. These parameters may be referenced in the command file as ~0, ~1, ~2,...~9. During command file processing, these parameters are replaced by their actual values,

taken from the invocation line.

The command definition file is prepared by the user with the editor. The default file extension may be .SUB and the default drive may be 0 for the command definition file. It may contain all printable ASCII characters plus space character and carriage return and linefeed. Five characters are given special treatment.

- linefeeds are ignored, carriage returns must separate each command line
- All dollar signs ($) are converted to an esc character for EDIT.CM.
- The tilde (~) is the command file character. It precedes command file keywords and the command file index.
- The percent (%) indicates a command file label. It is part of label references and definitions.
- The end of file (DC3) character must terminate the command file. The editor normally inserts this character into a command file.

A command definition file is assumed to have default extension SUB and exist on drive 0. Its contents may consist of

- MicroDOS commands or application program names
- responses to MicroDOS commands
- responses to application programs if they perform keyboard reads via READ of UT (for ex, EDIT, ASM8, MEM, PROM25)
- command file commands

4. Action

SUBMIT works in two phases. In phase 1, it reads and processes the command definition file creating an intermediate file named Z.TMP on drive 0. If the diskette in drive 0 is write protected or the drive is not active, Z.TMP is assigned to drive 1. Phase 1 occupies memory starting at #C000 and loads phase 2 code into memory #8A50-#8FFF. Phase 2 code also resides in #B2EB-#B440.

Phase 1:

- resolves parameters
- tokenizes command file commands
- converts $ character to esc character
- deletes ~COMMENT lines
- resolves labels and their references
- detects, reports and then aborts on errors

The final action of phase 1 is to set the high bit of the high byte of register E as the command file flag and to rewind Z.TMP for phase 2. Phase 2 is the runtime phase. Execution of intermediate file Z.TMP is performed. Phase 2:

Phase 2:

- substitutes READ of keyboard via UT with a read from Z.TMP
- executes all command file keyword commands
- detects, reports and aborts on errors
- detects error calls to CDERR to give the user a choice to either continue or abort
- sounds the bell character and exits to MicroDOS upon detection of end of command file (DC3)

Caution—do not use SUBMIT with the PLM compiler because both programs use the same memory space between #8A50-#8FFF. Some programs which do not use READ of UT will not work with SUBMIT (for ex. BASIC2).

SUBMIT files may be chained, but not nested. That is, SUBMIT may be the last command in a command definition file, but it may not appear in the middle of a command definition file.

A BNF (Backus-Naur Form) of the command file language is located in Appendix B. Below is a description and examples of command language. A carriage return (CR) delimits the end of a command line. A space delimits between parts of the command file line.

Expressions

All expresions consist of an operator between two operands. a single space delimiter must be present between operands and operator. The operands may be numeric constants, string constants, ~J index, or parameters. If a parameter is referenced as a string constant it must be enclosed in quotes. If the parameter is referenced as a numeric constant, no quotes are used. A numeric constant may be a maximum of 2 digits. A string constant may be a maximum of 12 characters in length, otherwise truncation to 12 characters occurs.

Only relational operators are permitted (=, <>, <, >, <=, >=). Only the ~IF command contains an expression.

Examples

```
53 <= ~J
'ANYSTRING' = '~0'
0 = 0
~1 > 0
```

All command file commands are recognized by their first unique characters. The possible command files commands are ~COMMENT, ~IF, ~GOTO, ~TYPE, ~LREAD, ~KREAD, ~SETJ, ~DECJ, ~INCJ, ~EXIT. They may be abbreviated respectively to ~C, ~IF, ~G, ~T, ~L, ~K. ~S, ~D, ~IN, ~E.

COMMENT

The ~COMMENT permits user annotation of the command file. These are especially useful for maintenance and readability reasons. The ~COMMENT lines are deleted by phase 1, so they do not appear in the intermediate file.

Examples

```
~COMMENT This file interfaces the
EDIT program to
~COMMENT automatically make
backups of files
```

~IF

The ~IF command permits conditional sequencing based on the evaluation of an expression. If the expression is found to be true the command file command following the expression is executed. Otherwise the next line is executed.

Examples

```
~IF ~J = 0 ~GOTO %LABEL1
~IF '~0' <> '' ~EXIT
~IF ~0 = 1 ~IF ~1 = 1 ~GOTO %L1
```

~GOTO

The ~GOTO command provides a means of altering the flow of command sequences. It permits a jump to a labeled line, either forward or backward. Labels must begin with a percent sign character (%). Labels are composed of a maximum of 9 alphanumeric characters following the percent sign. They are entered into a symbol table during phase 1 and used to resolve label references. At the end of phase 1, if any labels are not defined, an error message is issued and command file processing aborts.

Examples

```
~GOTO %BEGIN
~GOTO %ENDALL
```

~SETJ, ~DECJ, ~INCJ

The ~J index may be changed in value by operations to set it, decrement it by one, and increment it by one. ~J has a default value of 0, and may take on the range of values between 0 and 99. If ~J takes on a value less than 0, a phase 2 error message:

```
~UNFL
```

occurs. If ~J takes on a value greater than 99, a phase 2 error message:

```
~OVFL
```

occurs.

Example:

```
~SETJ 98
~INCJ
~DECJ
```

This sequence sets ~J to 98, increments it by one, and then decrements it by one. The final value of ~J is 98.

~LREAD, ~KREAD

The read commands permit pause for keyboard input during phase 2 of command file execution. ~LREAD permits a line of input terminated by a cariage return, while ~KREAD permits input until a termination keystroke (control d) is input. These features are useful for entering additional options at the end of a command line or to pause in mid execution to check for errors before proceeding.

Caution: ~LREAD and ~KREAD must be terminated by a (CR) in command definition file because phase 1 recognition ignores all characters beyone K or L until a (CR) is detected.

If the user wishes to use ~KREAD for a mid command line pause, he continuation of that command line must be on a new line.

Example

```
COPY ~KREAD
  DEST.FN
```

During phase 2, ~KREAD suspends execution so the user may enter via the keyboard the name of the source file which will then be copied to DEST.FN. Note the space needed before DEST.FN.

As another example, ~LREAD is used to permit completing options for the DIR command.

Example

```
DIR X. X;~LREAD
```

This example pauses for keyboard input to complete the options for the DIR command.

~TYPE

The ~TYPE command permits message display during execution of a command file. These messages may prompt the user for specific action during command file processing or simply report progress.

Example

```
~TYPE Please change disks in drive 1
   and type (CR) then ready
~LREAD
```

This sequence types a message to user to perform the action of a disk change and then pauses with the ~LREAD command, continuing after the (CR) character is keystroked.

~EXIT

The ~EXIT command directs that the command file is to be exited and control given to MicroDOS. No further commands are taken from the command file. This command can ensure that certain lines of the command file are not executed. For example if an error in handling routine is located at the end of a command file, an EXIT command would be placed preceding the routine:

Example

```
.
.
.
~EXIT
%ERROR
.
.
.
```

The ~EXIT command used in conjunction with the ~IF command is useful for providing more than one execution path in a command file:

Example

```
~IF '~0' = '' ~EXIT
~IF '~0' = 'TAPE' ~GOTO
  %TAPEIT
~IF '~0' = 'DISK' ~GOTO
  %DISKIT
~TYPE no valid device found
~EXIT
%TAPEIT ~COMMENT process
  tape file
.
.
.~EXIT
%DISKIT ~COMMENT process
  disk file
.
.
.
~EXIT
```

This command file tests a parameter for equality to the string value of null, TAPE, or DISK. If TAPE or DISK if found ~GOTO branches to the appropriate path for handling that type of file. The ~EXIT command before the label % TAPEIT ensures that commands after the label are not executed unless an explicit branch to that label is made. The ~EXIT command before the label % DISKIT serves the same function.

Limits

The limits of values allowed in command files are summarized below:

- ~J value range is 0 to 99
- Numeric constants may be only 1 or 2 digits, they are treated as decimal values
- String constants must be enclosed in quotes; maximum length is 12 characters
- Labels are preceded by the percent (%) character, followed by a maximum of 9 alphanumeric characters. The maximum number of labels is 10, otherwise the symbol table overflows
- Maximum number of parameters is 10. Parameters may be a maximum of 12 characters.

5. Error Messages

During phase 1, in most cases, when errors are detected an error message with a line number is displayed and command file processing is aborted. In two cases, however, warning messages are issued and processing continues. These cases are:

- when a null parameter value is found
- when string constants are truncated to 12 characters in length

During phase 2, two conditions may cause an abort:

- when the break key is depressed
- if a runtime error such as ~J value overflow or underflow, or bad expression

The format of a phase 1 error is a line number message followed by an error message. For example:

```
ERROR IN LINE NUMBER 00004
COMMAND FILE OPERATOR
  ERROR
```

Phase 1 error messages and some possible causes are detailed below:

CAN'T OPEN COMMAND FILE—command definition file not on default drive 0, does not have default extension SUB, or not given in invocation line

CAN'T OPEN COMMAND WORK FILE—insufficient space on diskette in drive 0, diskette not present in drive 0

CAN'T READ COMMAND FILE—attempt to read from command definition file fails

CAN'T REWIND COMMAND WORK FILE—attempt to rewind Z.TMP file at end of phase 1 processing fails

COMMAND FILE DUPLICATE LABEL—a second definition is found for a label already defined

COMMAND FILE KEYWORD PROBLEM—attempt to find end of ~KREAD or ~LREAD command fails

COMMAND FILE LABEL REFERENCE NEVER DEFINED—at end of phase 1, a label is found to be undefined

COMMAND FILE OPERAND ERROR—attempt to recognize an operand as a string constant, ~J index, or numeric constant fails

COMMAND FILE OPERATOR ERROR—operator not recognized, only <>, >, <, <=, >=, = are permitted

COMMAND FILE SYMBOL TABLE OVERFLOW—attempt is made to enter more than 10 symbols in symbol table

COMMAND LINE FILENAME ERROR—attempt to recognize command definition filename from invocation line fails

COMMAND LINE PARAMETER ERROR—parameter exceeds 12 characters in length, in the invocation line

EXPR SPACE DELIM NOT FND—a space delimiter is expected in expression but is not found

IMPROPER USE OF TILDE (~)—~ was not recognized to be part of command file command, or ~J, or parameter

INVALID STRING OPERATOR—operator found that is not <> or =

NUMBER EXCEEDS 2 DIGITS—numeric constants must be 2 digits or less

SETJ FOLLOWED BY A STRING EXPR—numeric constant must follow SETJ, but a string constant is found

TARGET OF GOTO NOT PRECEDED BY PERCENT (%)—expected label reference following a GOTO

not found
UNEXPECTED END OF FILE FOUND—a DC3 character found before logical end of command file found
Phase 2 error messages are as follows:
~CMD FILE ABORT—break key was detected
~EXPR ERR—operand other than ~J, numeric constant, or string constant found
~OVFL—~J exceeds 99 in value
~UNFL—~J below 0 in value
~SETJERR—value for ~SETJ does not evaluate to a numeric value
~ERR, TYPE Y TO CONTINUE>—call to CDERR detected, command file execution is suspended, user is given choice to continue or abort.

6. Examples:

Four examples follow illustrating how the command file facility may be used.

Example 1 contains simply the commands to MicroDOS that perform an assembly, creation of a binary file, and execution of the binary file.

The file EX1.SUB contains the following:

```
ASMB USEMAC.ASM,MAC.ASM,USEMAC.
  LST:1;M
CDSBIN USEMAC.LST:1,USEMAC.CM:0
USEMAC.CM
```

At invocation time the command line appears as:

```
SUBMIT EX.1SUB
```

Example 2 shows how parameters may be passed into a command file to allow varying source assembly and macro files to be assembled, made into binary files, and then executed. This command file performs the same sequence of steps as the one in Example 1 but it has the additional versatility that it may be used to assemble files other than nust USEMAC.ASM and MAC.ASM

In a file named EX2.SUB is the following:

```
ASMB~0.ASM,~1.ASM,~0.LST:1;
  M
CDSBIN ~0.LST, ~0.CM:0
~0.CM
```

The invocation line appears as:

```
SUBMIT EX2.SUB USEMAC,MAC
```

Example 3 shows a command file to automate use of EDIT.CM. It invokes the editor, specifies the input and output files, performs the appends to bring the file into workspace, and lists the first 22 lines. After the user completes his edit session by a control D deystroke, the command file performs an exit from the editor, creates a backup file, and renames the most recent output file as the most current version of the edited file. The user may think of his file as having a constant name.

In a file names EX3.SUB is the following:

```
EDIT
RO$$~0.~1:1
~0.TMP
AAB22T$$~KREAD
E$$U$$DEL ~0.BAK:1
RENAME ~O. ~1:1,.BAK
RENAME ~0.TMP:1,.~1
```

This file is invoked as:

```
SUBMIT EX3.SUB TEDIT,DAT
```

Example 4 illustrates use of the control structures and index. The parameter specifies the number of times the command file is repeated.

In a file named EX4.SUB is the following:

```
~SETJ 1
%START ~COMMENT this is a
   backup routine for disks
~TYPE Put a new diskette into drive
   1, type (CR) when ready to proceed
COPY F1.EXT:0 F1.EXT:1~LREAD
COPY F2.EXT:0 F2.EXT:1
COPY F3.EXT:0 F3.EXT:1
~INCJ
~IF ~J <= ~0 ~GOTO %START
```

Notice the placement of the ~LREAD command to insert a pause before the COPY command is completed with a (CR).

When this file is invoked as:

```
SUBMIT EX4.SUB3
```

Three backup copies of the specified files may be made.

1. Command: **SYSGEN**

2. Purpose:

SYSGEN is used to initialize new disks before they can be used by MicroDOS or to duplicate MicroDOS files from one diskette to another. It can be used to duplicate selective programs or entire diskettes to provide a backup copy. SYSGEN can be used to produce identical copies of diskettes or to produce the same information reorganized to eliminate file gaps that may have been generated during editing and program development. The reorganization will physically remove all previously deleted files and leave all unused sectors in one block rather than scattered throughout the diskette. This capability helps to compact data on the disk and frees up additional storage area. The system diskette should be in unit 0 and the new diskette in unit 1.

3. Format:

SYSGEN <DELIM> [;<OPTIONS>](CR)

<DELIM> is a command line delimiter, and

<OPTIONS> is one or more of the letters listed below with their meanings.

L List the file names being copied on the line printer

N Do not print the copied file names on the console or printer
O Copy the operating system
D Omit copying the operating system; retain existing DISK ID and directory on unit 1
E Make an exact copy of the diskette in drive 0. No file reorganization will take place. Every sector will be written exactly as it is on the disk in drive 0.

4. Action:

After the SYSGEN program has been loaded into memory and the directory has been loaded from the diskette in unit 0, the following message is printed.

INPUT USERID>

Up to 44 characters may be assigned to the USERID. This information will be placed in the ID sector. Whenever a DIR or FREE command is executed, the USERID will be printed.

After the user presses (CR), the following message is printed.

INPUT DATE (MM/DD/YY)>

Up to eight characters may be assigned to the date. No specific format is required for the date; the format shown is only a suggestion. This information will be placed in the ID sector. Whenever a DIR or FREE command is executed, the DATE will be printed.

After the date has been typed, the following message appears:

SELECT COMMAND-TYPE H FOR HELP

The user may type any of the following commands which will perform the prescribed function. Any command may be repeated any number of times.

HELP

Format: H(CR)
Action: The HELP command lists the format of the following commands and gives a short description of each command.

PRINT SELECTED FILES

Format: P(CR)
The P command will list on the console or line printer all the files from the directory of the diskette in unit 0 that are selected to be copied when the copy function begins.

PRINT NON-SELECTED FILES

Format: N(CR)
Action: The N command will list on the console or line printer all the files from the directory of the diskette in unit 0 that are not selected to be copied when the copy function begins.

QUIT

Format: Q(CR)
Action: When all commands finish executing, control is returned to SYSGEN
The QUIT command is used to return control to the operating system.

SELECT FILES TO BE COPIES

Format: S[<DELIM><FILENAME SEQUENCE>](CR)
<DELIM> is a command line delimiter, <FILENAME SEQUENCE>::=<FILE DESCRIPTORS>[,<FILE DESCRIPTOR>]n <FILE DESCRIPTOR>::=<FILENAME>/<FAMILY NAME>/<FILE NO.>/<<FILE NO.>-<FILE NO.>>
<FILE NO.> is the number associated with the file name from the listing produced by the PRINT or PRINT NON-SELECTED FILES command.
<<FILE NO.>-<FILE NO.>> includes all the files between these numbers for selecting files to be copied.

Action: After SYSGEN has begun, the first S or D command given will automatically perform the complement function for all file names not specified in the command. Each following S or D command, then, will perform only the explicit function. The select command will select all the files in the <FILENAME SEQUENCE> for copying.

DESELECT FILES TO BE COPIED

Format: D[<DELIM><FILENAME SEQUENCE >](CR) <DELIM> is a command line delimiter, and <FILENAME SEQUENCE> is a list of files to be deselected as described in the command "SELECT FILES TO BE COPIED".

Action: After SYSGEN has begun, the first S or D command given will automatically perform the complement function for all file names not specified in the command. Each following S or D command, will perform only the explicit function. The deselect command will deselect all the files in the <FILENAME SEQUENCE> from being copied.

REINPUT ID AND DATE

Format: I
Action: The following message is printed

INPUT USERID>

Up to 44 characters may be assigned to the USERID. This information will be placed in the ID sector. Whenever a DIR or FREE command is executed, the USERID will be printed.

After the user presses (CR), the following message is printed:

INPUT DATE
(MM/DD/YY)>>

Up to eight characters may be assigned to the date. No specific format is required for the date; the format shown is only a suggestion. This information will be placed in the ID sector.
Whenever a DIR or FREE command is executed, the USERID will be printed.

COPY COMMAND

Format: C(CR)

Action: The transfer of data will begin from unit 0 to unit 1 under the following conditions:

1. If no D or S command is executed, then SYSGEN will select all the files displayed in the P command for transfer. The disk will be reorganized with all free space in one block at the end of the disk.
2. If S*.* is typed, the result will be the same as in the previous paragraph.
3. If D*.* is typed, the output disk will have a blank directory with a copy of the operating system also on the diskette, if O (copy the operating system) had been typed as an option. Other wise, only a blank directory would be copied.

For all copies other than exact copies, file names will be printed on the console or line printer. If the operating system is to be copied, the user must be sure that the operating system is on the diskette in unit 0.

The BREAK key aborts the printing of file names or the copying of diskettes.

5. Error Messages:

If the diskette in drive 1 is write protected, the following will be printed.

THE DISKETTE IN DRIVE 1 IS WRITE PROTECTED

If after five attempts to read a track from drive 0 fail, the following will be printed:

TERMINATION ERROR WHILE READING FROM DRIVE 0

If after five attempts to write a track to drive 1 fail, the following will be printed:

TERMINATION ERROR WHILE WRITING TO DRIVE 1

In each case, control is returned to the operating system.

6. Examples:

Make an exact copy.

```
≥SYSGEN;E(CR)
IS IT OK TO COPY TO DRIVE 1?Y
   EXACT COPY BEING MADE
#Q(CR)
```

Reorganize files on a diskette containing an operating system. The system diskette in unit 0 must contain an operating system.

```
≥ SYSGEN ;0(CR)
TYPE USERID>XXXX(CR)
TYPE DATE>XXXX(CR)
#S*.*(CR)..Select all files to be copied
#C(CR)  ..Make a system diskette
IS IT OK TO COPY TO DRIVE 1?Y
#Q(CR)  ..Return to command interpreter
```

Reorganize files on a diskette not containing an operating system.

```
≥ SYSGEN(CR)
TYPE USERID>XXXX(CR)
TYPE DATE>XXXX(CR)
#S*.*(CR)
#C(CR)
IS IT OK TO COPY TO DRIVE 1?Y
#Q(CR)
```

Copy only the files having the extension CM from a diskette.

```
≥ SYSGEN(CR)
TYPE USERID>XXXX(CR)
TYPE DATE>XXXX(CR)
#D*.*(CR)..Deselect all files
#S*.*CM(CR)..Select all .CM files
#C(CR)
IS IT OK TO COPY TO DRIVE 1?Y
#Q(CR)
```

Initialize a new diskette to have a blank directory.

```
≥ SYSGEN(CR)
TYPE USERID>XXXX(CR)
TYPE DATE>XXXX(CR)
#D*.*(CR)..Deselect all files
#C(CR)
IS IT OK TO COPY TO DRIVE 1?Y
#Q(CR)
```

Add six files from 0 to existing MicroDOS diskette in unit 1.

≥ SYSGEN;D(CR)
#S 1-6(CR)..Select first six files to be copied
#C(CR)
IS IT OK TO COPY TO DRIVE 1?Y
#Q(CR)

1. Command: **TAPED**

2. Purpose:
TAPED is a copy routine that can take a data file from disk to cassette tape or from cassette tape to disk. It can copy ASCII only.

3. Format:
TAPED<DELIM><NAME1>
<DELIM><NAME2>(CR)
<DELIM> is a command line delimiter
<NAME1> is the name of the source file or source device, and
<NAME2> is the name of the destination file or destination device.
If <NAME1> is a disk file name, it is of the format
<NAME1>[.<EXTENSION1>][:< DRIVE1>]
and <NAME2> must be specified.
If <DRIVE1> is not specified, "0" will be used.
If <EXTENSION1> or <EXTENSION2> is not specified, blank will be used.
If <NAME2> is a disk file name, it is of the format
<NAME2>[,<EXTENSION2>][:<DRIVE2 >]

The following are mnemonics for the non-disk devices used with the command TAPED:
#TR Read from tape
#TW Write to tape

4. Action:
Two types of file copying can be reqested:
Disk to device
Device to disk

Disk-to-device copy is a transfer from a disk file to a cassette tape. Device-to-disk copy is a transfer from a cassette tape to a disk file.

To pause the transfer of the TAPED program, press the BREAK key on the keyboard. To abort TAPED after a pause, press the Q (QUIT) key. Any other key will continue the copying.

5. Error Messages:

<FILENAME> F.N. NOT FOUND	<NAME1> does not exist.
DIR FULL	No more room exists for another file name in the directory.
DISK FULL	No more room exists for file on disk. Some of the data may have been trans ferred.
INVALID FILE TYPE	Disk file being copied to a non-disk device has a file type other than ASCII or ASCII-HEX format. TAPED cannot dump non-ASCII files to an ASCII device. The Operating System or any operating system file cannot be copied.
INVALID DV	Disk was entered (e.g., #DK).
NO SUCH DV	Peripheral device specified does not exist in system.
INVALID DATA TRANSFER TYPE	Device requested does not transfer data in the direction requested (e.g., copy to an input-only device or copy from an output-only device).
COMMAND SYNTAX ERR	A name contained a wild-card construct, or no file name was found as the first or second parameter.

6. Examples:
Copy the ASCII file ASCII to the cassette tape.
≥TAPED,ASCII,#TW(CR)

Command: **U**

2. Purpose:
U is a utility program that allows restarting CPU execution at any specified address while MicroDOS is still in control.

3. Format:
U<DELIM>< ADDRESS>[<DELIM>
<PARAMETERS>](CR)

4. Action:
The program in memory located at the starting address specified will be executed. In addition, any specified parameters will be passed to the program being executed.

5. Error Messages:
None applicable.

6. Examples:
Restart UT71 at 8000H
≥U 8000(CR)

Provided that the Directory program has been loaded into memory, restart is giving the file name ABC:1;E as a parameter.
≥U,0000,ABC:1;E(CR)

1. Command: **VERIFY**

2. Purpose:
VERIFY compares two disk files. If any of the sectors do not compare, a message will be printed. If all sectors compare but one file is longer than the other, a message will be printed.

3. Format:
VERIFY<DELIM><FILENAM1>
<DELIM><FILENAM2>(CR)
<DELIM> is a command line delimiter.

If the extension for <FILENAM1> or <FILENAM2> is omitted, a blank will be assumed. If <DRIVE> is omitted for <FILENAM1> or <FILENAM2>, zero will be assumed.

4. Action:

When a successful verification has been completed, the following message is printed:

FILE #1 IDENTICAL TO FILE #2

The VERIFY command compares sectors between file 1 and file 2. If two sectors are not equal, the following message will be printed:

FILE #1 LSN XXXXX IS UNEQUAL TO FILE #2 LSN YYYYY

Verification will continue until the end of file is reached.

If the files are unequal in length, the following message will be printed:

FILE #X IS LONGER THAN FILE #Y

X, Y are either file 1 or file 2. Upon completion, control returns to the command interpreter.

If the files are of different types, they will not be compared and the following message will be printed:

MIXED FILE TYPES

and control is returned to the command interpreter.

Any time during the comparison, control can be returned to the command interpreter by pressing the BREAK key. The following message is printed:

ABORTED

4. User Program Generation

The user of the MicroDisk Development System MS2000 will generally be creating one of three types of programs:

1. A program designed to run on the MicroDisk Development System itself.
2. A program designed to run on a different CDP1802-based system, such as a Microboard system, but for which hardware is not yet available.
3. Same as 2, except that hardware is available and the program is to be downloaded into the hardware and tested.

In all cases, the original source file is created using the Editor. The file is then translated into machine code by use of ASM8 assembler or one of the optional compilers available. Finally, the program is loaded and tested. From this point on the operational procedure varies. Note also that the programs have to be molded to the hardware on which they are to be run.

The following paragraphs give a brief summary of the programming considerations for each of these three cases. Details on use of the programming and debugging tools of the MS2000 are given in subsequent chapters. For general CDP1802 programming information, refer to the **User Manual for the CDP1802 Microprocessor**, MPM-201.

Case 1

Programs designed to run on the MS2000 must adhere to the programming conventions of RCA software. Register assignments are:

R0 - Do not use; reserved for DMA operations.
R1 - Do not use; reserved for interrupt.
R2 - Points to a free byte on a stack; stack grows toward lower addresses.
R3 - Program counter.
R4 - Contains address of the CALL routine in UT71.
R5 - Contains address of the RETURN routine in UT71.
R6 - Points to a return point (or immediate byte) after a subroutine call.

The user should refer to the routine INIT1 and INIT2 described in the chapter on **Monitor Program UT71**, for aid in setting up the registers. He should also study the chapter on **MicroDOS User Functions**, to find out how to interface the MicroDOS operating system so as to be able to read and write disk files, input from the keyboard, and the like. At a more elementary level, the chapter on **Monitor Programs UT71** tells how to directly interface the Monitor Program UT71. Note that these functions require additional specific register assignments.

Programs planning to use MicroDOS functions must avoid the area where MicroDOS resides. Refer to the memory map given in Fig. 3. For similar reasons, a program cannot be loaded into the Utility Program's memory area.

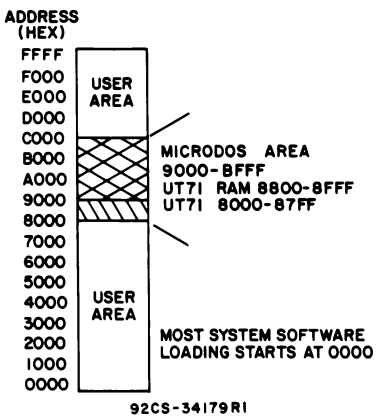


Fig. 3 - System Memory Map.

Once a program has been written and assembled, it can be loaded simply by typing its filename. Either a complete listing file, a hexadecimal-only file, or a binary file can be loaded this way. Unlike binary files, listing files may not begin to execute immediately. This delay is usually preferable during the debugging phase. The U command is used to start a loaded listing file.

Case 2

Programs intended to be run on a different CDP1802-based system and for which the specific hardware is not yet available can be loaded into the MS2000, and the terminal can be "borrowed" through interfacing with

MicroDOS user functions. Or, sections of code requiring no I/O can be tested in the System's RAM. The same considerations apply as for Case 1.

Case 3

A program designed for another system can be transported and debugged in one of two ways: (1) the program can be burned into PROM's, using a PROM programmer, or, (2) the program can be down-loaded into a RAM-based system using the MicroEmulator MSE3001 or the Micromonitor CDP18S030 and MOPS software. In either case, the MicroEmulator or the Micromonitor as a stand-alone device or, the Micromonitor in conjunction with MOPS, can be used for debugging.

5. Disk Editor

Introduction

The MS2000 Disk Editor (EDIT) is a program that facilitates the creation and modification of local files for storage on a floppy disk. Typically, the files are source programs. However, they may also be any other kind of conventional document.

After the user has written his assembly language program and wants to assemble and run it, he immediately faces the problem of converting the hand-written source file into a machine-readable form. This conversion involves a keyboard-to-disk operation in which lines on the coding sheet are transcribed to become lines on a source file. The Disk Editor will be used at this point to create the source file. The Editor provides assurance that the created files are in proper format for later reading by the assembler and for later modification, if necessary, by the Editor. Details on formats are given in the description of the Editor which follows.

Once a source file has been created and a first Assembly run made, it is very likely that error diagnostics will be returned by the Assembler asking for corrections to the source file to conform to its rules.

Typically, the changes required at this point are "trivial" but necessary. For example, spaces may have to be removed in one or more expressions. The same symbol may have been erroneously used for two purposes.An operation mnemonic may have been misspelled or a punctuation character such as a comma, colon, or single quote omitted. The number of possible trivial errors is clearly large.

To correct the errors and to alter the source file to conform the program to the Assembler's rules, the Editor is used. Typically, modifications at this point merely involve insertion and deletion of single characters or replacement of a small string of characters by a substitute string. The erroneous source file is used as an input to the Editor and the user generates a corrected source file as an output. The new file is then assembled or reassembled. At this point other trivial errors may appear that were not apparent on the first run. For example, an erroneous instruction operand may not have been flagged on the first assembly because its associated statement label or operation mnemonic may have also been in error. Thus, a new Edit-Reassemble pass may be necessary. Finally, a programs developed to which the Assembler does not object. At this point, a first run can take place.

The probability of a logical error in the program depends on its length and the previous experience of the programmer. Assuming one or more logical errors are found (via some "debugging" procedure), the source file must again be modified. Often such modifications are no longer trivial. For example, it may be necessary to find all instructions that branch to a given location and precede some of them with one or more instructions currently not in the program. Often, it may be necessary to delete some code or insert some code or move some code to a different point in the program. Several duplicated sets of in-line instructions may have to be removed and replaced with calls to one common subroutine which is to be added. The user may decide to "clean up" the program logically, in any one of several ways, or to improve its "readability" by modifying its comments or statement formats (by inserting TAB's or SPACE's, for example).

Such modifications to the source file also involve use of the Editor. After they are completed, a reassembly may again turn up new errors of the "trivial" variety. And so on. Thus the generation of a bug-free program typically involves the chart shown in Fig. 4. It is thus quite likely that the amount of time spent "conversing" with the Editor will be much larger than that spent with the Assembler.

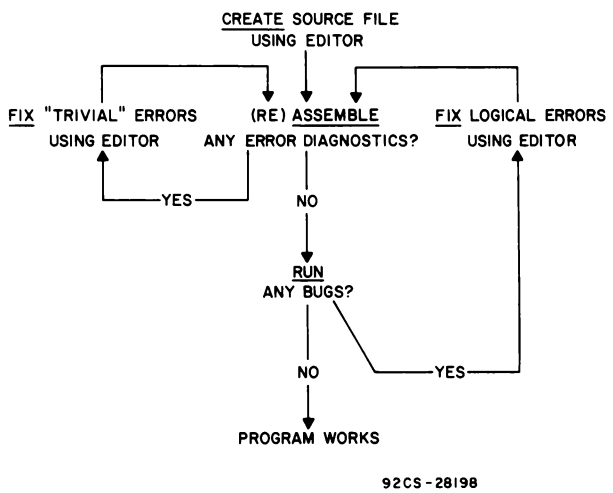


Fig. 4 - Flowchart for "bug-free" program.

A source program may be viewed as a long sequence of **characters**. When the Disk Editor Reads the source file, it places this character sequence in memory, with the code in each memory byte representing one source program character. The user is then free to type **commands** to the Editor to manipulate the memory representation of the program. For example, the user may identify a specific location and specify a character sequence to be **inserted** there. He may also identify certain characters to be **deleted** or **altered**. He may ask the Editor to **search** for the occurrence of specific character sequences, after which further memory modifications (corrections) may be made.(Details of available commands are given later).

After he is satisfied that the new memory representation of the file contains all of the desired changes (frequently the user begins an editing session with a hand-written list of the changes to be made), he asks the Editor to write (create) a new file containing the new version of the program. This new file is then used as the input file for a reassembly.

Operating Instructions

Memory Space Requirements

The EDIT program occupies approximately 6 kilobytes of memory space. It is supplied on the MicroDOS System Diskette for loading into the RAM of the MS2000.

EDIT requires about 100 bytes of the RAM work space for its own internal purposes. The remainder of the available RAM space is used as an editing area called a buffer. Virtually all EDIT operations involve the **buffer**. EDIT is designed to take advantage of all of the available RAM space below 8000H for its buffer area.

Input and Output Files

Normally, a user **creates** a file using EDIT by filling the buffer from the I/O terminal keyboard and then causing EDIT to write this information onto a diskette (which will contain the created file).

An existing (input) file may be modified (**edited**) by reading portions of it into the buffer, then using EDIT commands to alter the contents of the buffer, and finally writing the results onto the output file. Typically, the output file is a new version of the input file. After an editing session, the new version is retained and the old version is discarded (although it may be temporarily saved for future reference or backup).

Thus, EDIT has means to read an input file into the buffer, means to examine and modify the contents of the buffer in many ways, and means to write the buffer contents onto an output file. Alternatively, when an input file does not exist, the user creates an output file by loading the buffer from the keyboard.

Record Formats

In order to understand the various commands EDIT is designed to execute, it is fundamentally important that the user understand how information is normally recorded on the disk and in the buffer.

A file is a sequence of records or **lines**. Each line consists of a sequence of **characters**. The length of a line is restricted to 78 or fewer characters of data. Thus, a line in a file is normally printable as a line on the I/O terminal printer. Each character is represented by an 8-bit ASCII code or byte, either on the file or in memory. Typically, every character in a line is a **printable** character (including space or blank). Every nonprinting character code represents a **control** character. A control code may be generated on the keyboard either by hitting an appropriately marked key (e.g., RETURN, ESC, etc.) or by depression of the CTRL button while hitting another key. The terminal reacts to the receipt of a control character in one of several possible ways. Some control characters (such as carriage return, line feed, bell, etc.) cause the terminal to execute a specific control function. Other control codes either are ignored by the terminal or may generate a special symbol on the display.

A line in a file may contain control characters (with certain restrictions to be discussed later). EDIT treats most of the control characters it encounters within a line in the same manner as it treats printing characters. However, certain control characters have special meaning in EDIT.

The proper format for disk files is shown in Fig. 5. Each line is terminated with a CARRIAGE RETURN (CR), and an optional LINE FEED (LF). Note that the last line of the disk should be followed by a "dummy" line containing only the single character DC3. DC3 is a special control character generated on the keyboard by hitting CTRL and S. It acts as an END OF FILE indicator.

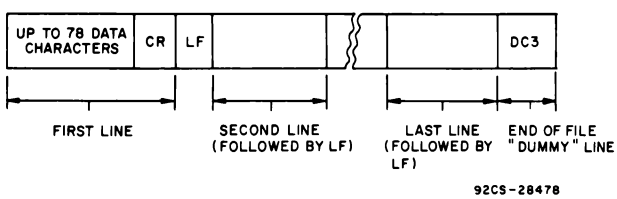


Fig. 5 - Disk file format.

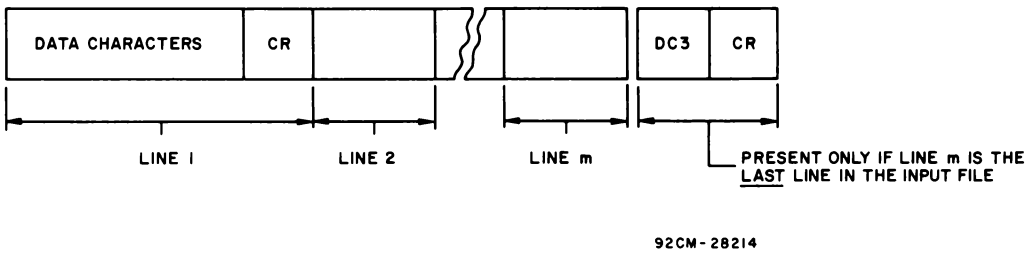


Fig. 6 - Memory buffer format.

File records read by EDIT are deposited into the buffer as they appear on the diskette, but with all LF's ignored. While EDIT operates on the data in its buffer, it specifically uses the CR character as an indicator of the end of a line. (Recall that a line has a variable length.)A new line is assumed to start with the next character in the buffer. Thus, the buffer format is of the form shown in Fig. 6.

When EDIT is depositing keyboard data into its buffer, the ASCII code equivalent of each struck key (any printing character and almost any control character, with exceptions as noted below) goes into memory and is also "echoed" back to the printer. EDIT, however, especially ignores the LF key. Further, when the RETURN key is hit, the CR character goes into memory and a CR, LF **pair** of characters is echoed back to the printer to start a new line. Thus, the user terminates a line of keyboard input with a single carriage RETURN. Normally, then, the LF character should not appear at any point in the buffer.

Whenever EDIT transmits a CR character to the terminal, it automatically appends to it LF and NULL characters to provide sufficient time delay for the carriage to settle.

It is conceivable that because of a user error, one or more lines on the input file or in the buffer may exceed the 78 data character length restriction. For example, data alterations in the buffer may have resulted in deleted CR's. (Note that each CR deleted in the buffer causes the **concatenation** of its adjacent lines.) EDIT has the following provisions for handling lines that exceed the length restriction:

(1) Whenever EDIT is outputting a line to the terminal as the result of a user TYPE command, if the line exceeds 78 characters, a "LINE TOO LONG" message will also be printed.

(2) If EDIT encounters too long a line while writing from the buffer to the disk, the line will be broken up, using as many 78-data character records as are necessary each terminated by a CR.

(3) A line which is too long on the input file is truncated to 78 characters, with a CR appended, in the buffer.

Buffer Pointer

The total RAM space available for the buffer is generally partially filled. When EDIT is first initialized, the buffer is empty. When data is added to the buffer (from the keyboard or from the disk input file) the buffer expands. When data is deleted, the buffer contracts. EDIT continually keeps track of the present extent of the buffer within the work space.

EDIT maintains a virtual **pointer** which identifies some point **between** two characters in the buffer. This pointer has the same function as what is commonly called a "cursor". Most EDIT operations are executed relative to this pointer. Further, several EDIT operations exist specifically to alter the location of the pointer. Because the pointer is not visible, it is the user's responsibility to keep track of where the pointer is. Often, its location is verified by asking EDIT to type information in the buffer at the current pointer position. Alternatively, the user may first initialize the pointer to a known reference point (e.g., the beginning or end of a line, or the beginning or end of the buffer) and then move it relative to this known origin.

In illustrative examples, the location of the pointer is indicated with an arrow below and between the two buffer characters. For example, in

AB CDE
↑

the character before the pointer is B and that after the pointer is C.

Unless otherwise noted, whenever text is deleted from the buffer, the character sequence to be deleted exists either immediately to the right or immediately to the left of the pointer. After the deletion, the buffer has contracted by the number of characters deleted. If the field deleted is to the right of the pointer, the character immediately to the left of the pointer remains the same. The character to the right of the pointer then becomes the character that was immediately to the right of the deleted field. A corresponding statement can be made for deletion to the left of the pointer.

When text is inserted, the buffer expands. Unless otherwise noted, text is inserted between the two characters at the position of the pointer. After the insertion, the

pointer is positioned immediately after the inserted test. Thus, the character to the right of the pointer remains the same.

The execution of many EDIT operations starts at the present pointer position and proceeds either towards the end or towards the beginning of the buffer. EDIT insures that the pointer cannot be moved past the present limits of the buffer. If the pointer reaches the beginning or the end of the buffer, the operation stops -leaving the pointer at that point. For example, if the pointer is positioned n characters from the end of the buffer and the user asks to move the pointer m characters to the right, with m greater than n, then the operation will stop after the buffer pointer has been incremented by only n.

EDIT Command Operation

Command Strings

When control is transferred to EDIT, it will print the initial message

COSMAC DISK EDITOR VER.X.XX

and then follow this message with its "→>" user prompt. The →> prompt always indicates that EDIT is ready to receive a new user command from the keyboard (having executed the previous one).

After receiving the →>, the user types a sequence of one or more commands which EDIT will execute in order. The first command should tell EDIT where to read the input file and where to write the output file. (See later discussion of **EDIT File Assignments.**) Most commands may be optionally delimited (ended) by an ESCAPE character. Commands which include text arguments of variable length must include this character to define the end of a text field. The command string is **always** terminated by **two** successive ESCAPES.
Because the (CR) character (often used as a line terminator) is treated by EDIT as data, it cannot be used as the command terminator. EDIT uses instead the **ESCape** character.

The system operates in the full duplex mode. Normally, a program merely "echoes" back to the display which it has just received from the keyboard. However, whenever EDIT receives an ESC character, it is echoed back to the display followed by a $ to give a visual indication of the ESC key depression. Thus, a typical command string normally appears on the screen as

COMMAND1$COMMAND2$...
COMMANDn$$

where in most cases the separating ECS's are optional but the final pair is mandatory. A command string must be terminated by two depressions of the ESC key.

Command Formats

The heart of the command is a single letter mnemonic (such as "T" for TYPE, "I" for INSERT, etc.). In many cases, this letter may be optionally preceded by a decimal number (later denoted by n) indicating the number of characters or lines involved. Further, in some cases this number may be preceded by a minus sign (-) indicating a direction (from the present pointer position) toward the beginning of the buffer rather than toward the end (as is normally assumed). If no number is present, EDIT assumes the value 1.

Given an arbitrary pointer location, the possible EDIT interpretations for n are normally as follows:

(1) **Character Operations:** Positive n identifies the n characters to the right of the pointer (including control characters and spaces).
Negative n identifies the n characters to its left. Unless otherwise noted n=0 results in no operation.

(2) **Line Operations:** Positive n identifies all characters to the right of the pointer up to and including the nth CR encountered. If the pointer is in the middle of a line, the first line will constitute only the **remainder** of that line. Negative n identifies all characters to the left of the pointer up to but not including the -n + 1st CR. If the pointer is in the middle of a line, the last line (in this set of lines) will consist of only those characters in the present line to the left of the pointer. Thus, n=0 specifically indicates the portion of the present line to the left of the pointer.

In certain cases a command mnemonic letter is followed by one or two variable-length text arguments (whenever the user needs to specify some sequence of characters to insert or to search for). All such arguments must be terminated by the ESC character (echoed as $). In subsequent discussion, an arbitrary text argument will be denoted by a symbolic statement such as "text".

Correcting Command Typing Errors

A typing error in a command string may be corrected by use of the RUBOUT (DEL) character to 'erase" previous characters already typed. Each time EDIT receives a RUBOUT within a command string, it erases the last character from its stored version of the command string. Further, it echoes back to the terminal the character just erased. For example, suppose the user types the command string ABS$DE (each of the letters

is a valid command mnemonic) followed by four rubouts. On the terminal, he would see

where the last four characters were those erased. The characters AB would then remain in EDIT's stored command string register. Clearly, any such erasures must occur before the double ESC character, which terminates the command string, is struck.

If EDIT finds an invalid command while in execution of a command string (i.e., after the user has typed the double ESC), it returns to the user the error message

BAD COMMAND??"xxxx..xx$"

where xxxx..xx reproduces that part of the command string that has not been executed.

Interrupting EDIT Execution

The user may usually stop EDIT execution by depressing and holding the BREAK key on the keyboard. This key is used, for example, to stop a long typeout. On receipt of the BREAK, EDIT stops execution at whatever point was reached and returns to the command input mode by issuing another prompt. To assure the clean entry of succeeding commands, the DEL key should be depressed to erase any erroneous noise characters that may have been entered as a result of the break.

After a BREAK, the user should normally verify or reinitialize the buffer pointer position before resuming further editing.

Filled Work Space Warning

If EDIT determines that a command string threatens to use up the remaining work space, it will stop echoing keyboard input characters to the printer and will echo instead the BELL control character causing the I/O data terminal to ring its bell as a warning. The user should immediately respond by erasing part of it with the RUBOUT key until the bell stops echoing. It is particularly important during an INSERT that when the bell sounds, additional characters are not entered. The last few characters of the buffer should be deleted and the INSERT mode ended. After some of the buffer is written out, the user should go back and repair the last line as necessary. An attempt to insert more characters after the bell can result in the loss of the entire buffer contents. The WRITE AND DELETE command W is used to empty the buffer onto the diskette.

If the EDIT runs out of space during command execution, it will return the error message

MEMORY FULL"xxx..xx$"

where again, xxx...xx is a reproduction of the unprocessed part of the command string.

File Assignments

The Editor program is loaded by means of the command interpreter. Output generated by the program is underlined. The $ symbol indicates the ESC key.

≥ EDIT
COSMAC DISK EDITOR VER X.X
→≥

At this point EDIT is asking the operator to assign an input file and output file. A new file name can be established during the course of an EDIT session without having to restart the EDIT program. The new file can be established any time after a →> is received. Each time EDIT is restarted, via the E, Y, or Q commands (explained in the next section **EDIT Commands**),the output and input files are closed. The format for input and output file name assignments is shown below.

→>R$$
READ = <FILENAME>(CR) ..Default unit No. is 0

→≥O$$
WRITE = <FILENAME>(CR) ..Default unit No. is 1

→≥

Note:The R and O commands may be issued at the same time as shown below.

→≥RO$$
READ = <FILENAME>(CR)
WRITE = <FILENAME>(CR)

→≥

EDIT Commands - Single

This section contains a summary of the individual commands that EDIT is designed to recognize. Each command is described with a specification of its acceptable format and an explanation of its execution. Examples are also given.

Pointer Control Commands

BEGINNING

Format: B
Execution:Pointer repositioned to the beginning of the buffer.

END OF BUFFER

Format: Z
Execution:Pointer repositioned to the end of the buffer.

Note:This command must be used with caution. If the current buffer contains a DC3, the Z command will reposition the pointer beyond the DC3. Any insertion made after the Z is typed, consequently, will not be added to the file because it is beyond the DC3 or end-of-file. The user, therefore, should always type Z-L to position the pointer in front of the DC3.

CHARACTER STEP

Format: nC
Execution: Step pointer right (or left¢) by n characters.

LINE STEP

Format: nL
Execution:Step pointer down (or up) by n lines.

TYPE LINE NUMBER

Format: *
Execution: Type line number of pointer position within buffer.

File Manipulation Commands

INPUT FILE SELECTION

Format: R
Execution: Causes EDIT to type

READ = <FILENAME>

The operator should respond with the file name of the source file. This command may be issued at any time during the edit session. This command opens a file for reading. If the R command is entered by mistake, the operator should respond to the "READ=" prompt with an (ESC) or (CR). The read assignment will be unchanged.

OUTPUT FILE SELECTION

Format: O
Execution: Cause EDIT to type

WRITE = <FILENAME>

The operator should respond with the file name of the source file. This command may be issued at any time during the edit session. If the O command is entered by mistake, the operator should respond to the "WRITE=" prompt with an (ESC) or (CR). The write assignment will be unchanged. This command opens a file for writing.

¢ A positive (unsigned) n indicates the direction of right or down; a negative n indicates left or up for all commands.

APPEND

Format: A
Execution: Lines are read from the input file (continuing from the last line) and appended to the **end** of the buffer. The operation continues until one of the following occurs:

(1) End of file character detected (i.e., last line has been read).
(2) 3/4 of the remaining available space has been filled.
(3) 100 lines have been transferred.

The pointer is repositioned to the beginning of the first appended line. In large memory systems, multiple appends may be used to bring additional lines into the buffer. Multiple APPEND commands must be typed as a string of A's. The form nA is not acceptable.

Note: The keyboard BREAK key should not be used during execution of an APPEND.

NEXT

Format: nN
Execution: Lines are read from the input file (continuing from the last line) and appended to the **end** of the buffer. The operation continues until one of the following occurs:

(1) End of file character detected (i.e., last line has been read).
(2) 3/4 of the remaining available space has been filled.
(3) n lines have been transferred.

MERGE FILE

Format: M
Execution: Allows further appends to the buffer. The difference between this command and an A(ppend) command is that once an end-of-file marker (DC3) is read, EDIT will not allow further Appends until this M(erge) command has been issued. After the M command is issued, the Append is used to bring subsequent sections of a second file into memory. When this command is used, it is assumed that the end of the current file is already in the buffer. To merge this file with another one, the user must first delete the end-of-file marker from the buffer, select the input file to be merged, and then issue the M command. An example is given below. Output generated by EDIT is underlined.

-> A$$	..Bring the end of the current file into memory.
EOF	..Editor responds that the end-of-file has been reached.
->	
-> BF(DC3)$-D$$	..Find and delete the ASCII (DC3) control ..character end-of-file marker

```
→> R$$          ..Select file to be merged
READ=XY2(CR)
→> M$$          ..Enter the first portion of the
                  new file at the end of the memory
                  buffer
```

Repeated A100W$$ commands are then issued until the next EOF is found. The second file is now following the first.

Deletion Commands

DELETE

Format: nD
Execution: n characters right (or left) adjacent to the pointer are deleted.

KILL

Format: nK
Execution: n lines right (or left) adjacent to the pointer are deleted.

Text Insertion and Data Manipulation

INSERT

Format: Itext$
Execution: Typed text is inserted to the left of the present pointer position. The text may contain multiple lines.

SAVE

Format: nX
Execution: Copy n lines adjacent to the pointer into a special SAVE area external to the buffer. The pointer position is not changed. Previous contents of the SAVE area are overwritten. EDIT types CAN'T SAVE if there is insufficient room in the SAVE area and it does not save any lines. EDIT clears the SAVE area if n=0 (zero).

GET

Format: G
Execution: Equivalent to a INSERT, but uses the present contents of the SAVE area as an implicit text argument. Note: SAVE and GET are especially useful in sequence as a copying mechanism to MOVE text.

EDIT dynamically allocates the available RAM work space to its SAVE area, stack area, and the buffer or editing area. Once lines have been SAVE'd, they remain in the SAVE area indefinitely until the next SAVE command overwrites them. If many characters have been SAVE'd, the area available for the buffer will be proportionally reduced. The SAVE area is not automatically cleared by a GET command. Several GET commands may be issued against the same SAVE area. It is good practice, therefore, to clear the SAVE area when it is no longer needed in order to make that area available to the buffer. This step is accomplished by typing 0X (zero-X).

If an attempt is made to save more lines than there is room for, EDIT will type

```
CAN'T SAVE "XXXX...XX$"
```

and will not transfer any lines to the SAVE area. XXXX...XX is the portion of the command not executed.

FIND

Format: Ftext$
Execution: A search for the specified character sequence "text" occurs from the current pointer position toward the end of the buffer. It stops either when a match is first encountered or when the end of the buffer is reached. In the first case, the pointer ends positioned immediately **after** the matching string. In the latter case, a "CAN'T FIND" message is printed, and the pointer position is unchanged.

SUBSTITUTE

Format: S search text $substitute text$
Execution: Operates as FIND does above (using search text as the search argument). However, on a match, the substitute text **replaces** the matching sequence with the pointer positioned **after** the inserted text. The substitute text must not be omitted from the command.

Output Commands

TYPE

Format: nT
Execution: Type the n lines adjacent to the current pointer. The pointer position remains unchanged.

PRINT

Format: nP
Execution: The n lines adjacent to the pointer are sent to a printer or punch if one is provided. The pointer position remains unchanged. The lines are not deleted from the buffer.

TYPE EDITOR STATUS

Format:#
Execution: Type out size of the buffer, number of bytes available, size of the save area, and the end of memory.

WRITE and DELETE

Format: nW
Execution: n is treated as **positive**. The n lines at the beginning of the buffer are written to the output file and

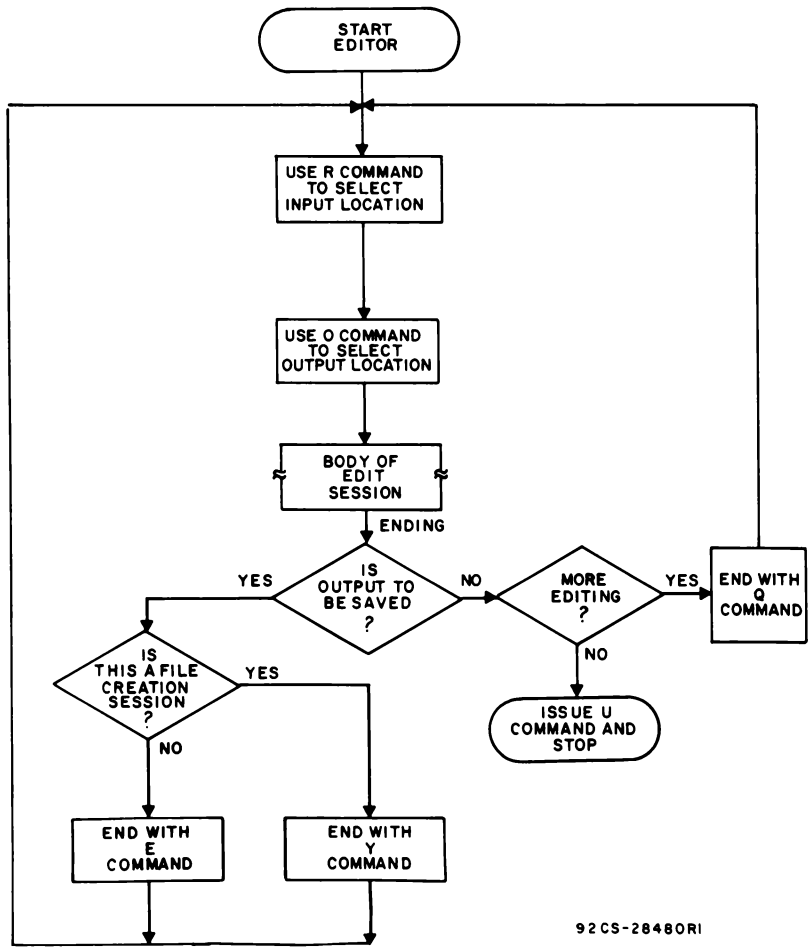


Fig. 7 - Flowchart showing methods for terminating an EDIT session.

deleted from the buffer. The pointer ends up positioned at the beginning of the remaining buffer.

END

Format: E
Execution: The buffer is written to the output file and any lines remaining on the input file are then copied to the output file and the file is closed. EDIT then reinitializes for a new editing session with buffer cleared and with the pointer positioned at the beginning of the work space.

FILE CLOSE

Format:Y
Execution: Places an end-of-file character (DC3) at the end of the working buffer, outputs the buffer to disk, and restarts EDIT. ALL FILE CREATION SESSIONS MUST END WITH THIS COMMAND. Fig. 7 shows the methods of terminating an edit session. The Y command may also be used to truncate a copied file.

QUIT EDIT SESSION

Format: Q
Execution: Restarts EDIT. Execution of this command destroys the contents of the working buffer. Fig. 7 shows alternate methods of terminating an edit session. The output file is not closed.

RETURN TO UTILITY PROGRAM

Format: U
Execution: Restarts CDOS, which will type a > to the terminal indicating that it is ready to accept commands. No closing of file will take place.

Summary of Commands and Control Characters

A summary listing of the foregoing commands together with the meaning of each one is given in Table III. A summary of the special EDIT control characters

Table III - EDIT Command Summary

Format	Meaning
R	Define input (Read file name). Response READ=FILENAME
O	Define output file name. Response WRITE=FILENAME.
B	Move pointer to BEGINNING of buffer.
Z	Move pointer to END of buffer.
nC	Step pointer right (or left) by n CHARACTERS.
nL	Step pointer down (or up) by n LINES.
*	TYPE out the line number of the pointer within the buffer.
A	APPEND lines to end of buffer from input file. Reposition pointer to beginning of APPENDed area.
nN	APPEND the next n lines into the buffer, if there is room. Default for n is 1.
nD	DELETE n characters after (or before) pointer.
nK	KILL n lines after (or before) pointer.
Itext$	INSERT text at present pointer position. (Position pointer after it).
nX	Save n lines after (or before) pointer. (Pointer position unchanged.)Clears the SAVE area if n= 0.
G	GET the last SAVEd lines and INSERT them.
Ftext$	FIND the first occurrence of text, searching from present pointer position toward end of buffer. If found, position pointer after the match. If not, type CAN'T FIND.
Ssearch text $substitute text$	FIND search text and SUBSTITUTE substitute text for it.
nT	TYPE n lines after (or before) pointer. (No change in pointer location.)
nP	PRINT/PUNCH n lines after (or before) pointer. (Buffer and pointer remain unchanged.)
nW	WRITE (and delete from buffer) the first n buffer lines on the output file. n is positive. (Pointer ends up at beginning of remaining buffer.)
#	TYPE Editor status.
E	END the editing session. Equivalent to an nW, with n equal to or greater than the number of buffer lines, followed by a copy of remaining input file to output file.
Y	Used to end a file-creation session. Places an end-of-file marker on the bottom of the buffer and outputs the buffer.
Q	Restart Editor program and clear buffer.
M	Merge buffer contents with selected input file.
U	Exit to MicroDOS.

is given in Table IV. The EDIT error messages are summarized in Table V.

The EDIT error message

```
DISK FULL
  SET UP CONTINUATION FILE
  WRITE?
```

is of interest because it tells the user how to proceed. The user should replace the full disk with one that has free space and then enter the continuation <FILENAME> after WRITE? The remaining output will be stored under this file name. Caution must be exercised, however, when disks are being changed that the source input is not removed. This file continuation procedure can be used any number of times. Before anything is done with the output files, however, they must be merged by means of the CDOS MERGE command. MERGE is the only program that can accept multi-file inputs.

EDIT Commands - Composite

EDIT also permits the user to specify **composite** commands. A composite command is a command string (one or more commands) enclosed within angle brackets (<...>). A command string may be preceded by a decimal number indicating the number of times that the string within the brackets should be executed.

Table IV - Summary of EDIT Control Characters

Message	Meaning
(1) ESCAPE	Echoed as $. Optional command separator. Required after a TEXT field. Two required at the end of a command string.
(2) LINE FEED	Ignored on input. Inserted after CR on output.
(3) CARRIAGE RETURN	Line terminator character. Stored in buffer.
(4) NULL	Ignored on input. Set of six inserted after LF to terminal
(5) RUBOUT or DELETE	Erases previous character in a command string.
(6) DC3	End-of-file character. Inserted by user at end of a created file or read in from an existing input file.
(7) HORIZ TAB	Echoed as 1 to 8 spaces when typed. Converted to 1 to 8 spaces on file output. Can begin a command implying a previous INSERT.
(8) BREAK	Pressing BREAK will terminate a long command.

Note: Within a command string but not within a text field, EDIT ignores any inserted spaces or CR's. Spaces or CR's may be used to improve the readibility of the command string if desired.

Table V - EDIT Error Messages

Message	Meaning
LINE TOO LONG	A line that EDIT is attempting to TYPE has more than 78 characters.
BAD COMMAND?? "XXX..X$"	EDIT has found an invalid command in a command string. XXX...X is that part of the string not executed.
<BELL>	Filled work space warning. Delete part of the command before ending the command.
MEMORY FULL "XXX..X$"	EDIT ran out of work space during an execution. XXX..X is the unprocessed part of the command string.
CAN'T SAVE	There is not enough room in the SAVE area.
CAN'T FIND "text"	The specified character sequence was not found between the pointer's previous position and the end of the buffer.
<XX> IS WRITE PROTECTED	The disk unit selected (XX) for output is write protected. The command string is aborted. No lines are written or lost.
<XX> DR FAIL	The disk unit selected for output is not ready. The command string is aborted. No lines are written or lost.
ITERATION STACK FAULT	EDIT ran out of stack space during execution of a command string. May indicate improperly paired brackets in the string.
EOF	A line containing an end-of-file mark (DC3) has been read. The DC3 is stored in the buffer and further appends from the current file are ignored.
DISK FULL SET UP CONTINUATION FILE WRITE?	Output disk full. Replace disk and enter continuation file name after the query WRITE?

One composite command may include another. Thus, EDIT permits the "nesting" of commands. For example.

B5<3C4<D1$>L>$$

causes replacement of the 4th through the 7th characters in the first 5 lines in the buffer by spaces. The pointer ends positioned at the beginning of the sixth line.

With nested commands, the user must be aware of the order in which commands will be executed and the number of times individual operations will occur. The

following example should indicate the general algorithm. Other examples will be given later. Consider the command string

a<b<CS1>c<d<e<C S2>CS3>CS4>>

where the lower case letters represent numbers and where each CSi represents an elementary command string. Fig. 8 indicates EDIT's flow chart for the execution of this command string. It is derived by properly pairing the angle brackets in the string.

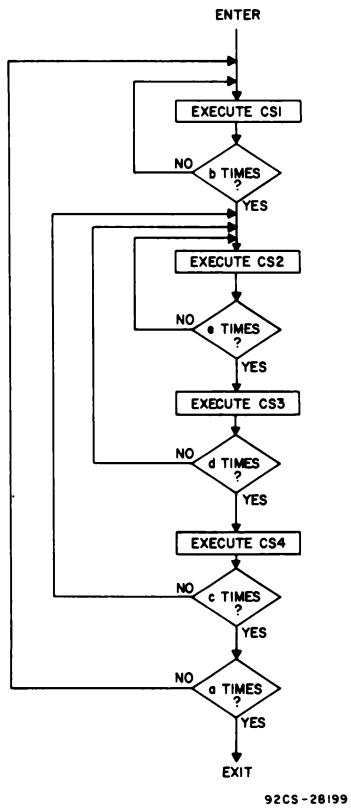


Fig. 8 - Execution of nested composite commands.

Notice, for example, that CS2 is executed a number of times equal to the product of a, c, d, and e.

To execute a nested command, EDIT maintains a stack in part of the available work space. The amount of stack space required depends on the depth of nesting in the command, i.e., on the number of loops within loops, as in Fig. 8, which in turn depends on the depth of bracket-pairs-within-bracket-pairs in the command string. If EDIT runs out of stack space during execution, it will issue the error message:

ITERATION STACK FAULT.

This error message is most likely to occur if the brackets in the command string are not paired properly. In particular, it occurs if a bracket is missing.

Note that if the user fails to terminate a text string with the required ESC character, all subsequent characters until an ESC **does** occur will be treated as part of the presumed text string. Thus, it is quite possible that a missing ESC in a nested command string could also result in the "improperly paired-brackets" error message

ITERATION STACK FAULT.

Horizontal Tabs

EDIT assumes an implicit horizontal tab stop after every **eight** character positions in a line. If the user types a HORIZ TAB character (CTRL and I) as part of a text field, EDIT will insert this character into its buffer, but it will echo back to the printer a sufficient number of spaces to reach the next implied tab stop. HORIZ TAB characters read from the input file are loaded into the buffer as is. On output, each HORIZ TAB buffer character is converted into the required number of spaces, extending the line length in the process. Thus, HORIZ TAB characters cannot appear on the output file. The TAB character can be used to produce straight columns in a source file.

NOTE: As a special case, EDIT interprets a text beginning with a HORIZ TAB character as if an INSERT command had preceded it.

Additional Note

Normally, the INSERT of a non-existent text field (i.e., the command I$) results in no operation. Further, it is normally illegal to precede an INSERT command with a numeric argument. However, the specific command nI$ (combining the two), is legal. It causes the insertion of a single character whose ASCII decimal value is n(modulo 128). For example, 97I$ will cause insertion of an "a" (hex 61).

File Development and Manipulation

In this section, information is given on the development and manipulation of a file through the use of the EDIT. In addition, some useful common sequences are given to illustrate EDIT's data manipulation facilities.

Creating a File

A file is created by a repeated sequence of the following steps:

(1) Fill buffer from keyboard with sequence of INSERT's
(2) WRITE buffer to output file.

A single I command may take as an argument a text string of arbitrary length. Thus, many lines may be inserted with a single I command. Each line is terminated by pressing the RETURN key. A typical INSERT will thus appear on the printer as

```
I line 1
  line 2
  .
  .
  .
  line n$$
```

because each CR is echoed as CR, LF. Such commands may be sequenced until the buffer is nearly filled. These sequences are then normally followed by an nW (WRITE) command with n equal to or greater than the number of lines in the buffer. By use of the W command, the buffer is cleared after the WRITE to the output file and is ready for a new set of INSERT's.

The last line of a created file should be followed by the insertion of a terminating dummy line consisting of the single character DC3 (CTRL and S) indicating the end of the file. The DC3 character is automatically added when the Y command is used to end a file-creating session. The file-terminating commands Y and E also generate a string of null characters after the DC3 to assure that data is written on the diskette.

Adding to a File

A section is added to an existing file by first copying the portion before the insert and finally copying the portion after the insert. The first copy involves one or more APPEND's followed by WRITE's up to the APPEND which reads in the section of the input file containing the insertion point. Note that appending to the end of a file may also be considered as an insertion just before the last DC3 terminating line.

Assuming the insert point is arbitrarily located within the buffer, several variations exist for adding text material. For any of these variations, the pointer must first be moved to the insert point. Then a sequence of INSERT's is made at that point, particularly if the amount of the inserted material is small. Alternatively, one could SAVE all lines following the pointer (with an nX, n sufficiently large), delete them with an nK command, and then WRITE the data remaining in the buffer with an nW (n sufficiently large). The buffer then becomes empty with all records preceding the addition written to the output file. Additional INSERT's and WRITE's may now be made. Finally, a GET followed by a WRITE will attach the material after the insert point. Now, if there is more unread material on the input file, the GET may be followed directly by an END command. This command will automatically copy the remaining input file.

In summary, one inserts material into an existing file by beginning with a copy sequence (a series of APPEND's followed by WRITE's). Then, with the pointer positioned properly, one may execute nX nK nW (n sufficiently large). Now, one operates in the CREATE mode with INSERT's followed by WRITE's. Finally a GET or GnW will complete the sequence.

When appending to the **end** of a file, one has the alternative of removing, after the last APPEND, the dummy termination line via a Z-IK command string. Operation then is as in the CREATE mode. For this case, the Y command should be used to terminate the file.

Deleting a Section in a File

To delete a section in a file, the user should first copy up to the deletion point, as previously discussed. Lines to be omitted may then be explicitly deleted from the buffer (by nK, with pointer properly positioned). If further lines to be deleted exist on the input file, further APPEND's are required.

Moving a Section in a File

Assume that the file section to be moved is sufficiently small. If the movement is toward the end of the file, the following sequence may be used:

(1) Copy input file up to the section to be moved.
(2) SAVE the section to be moved. Then DELETE it in the buffer.
(3) Continue copying the input file up to the insertion point.
(4) GET and WRITE the SAVE'd section.
(5) Copy the remaining part of the input file.

If the movement is toward the beginning of the file, one must first find the section to be saved, SAVE it, DELETE it, and then **reinitialize** the input file. After this, the sequence of steps 3, 4, and 5 above will effect the insertion.

Several complications of this simple procedure can occur. First, the material to be moved may overlap two APPEND's. In this case, one does not SAVE until the second APPEND has been executed. Second, the material to be moved may consist of a substantial portion of the input file so large that it must first be copied on to a third temporary file which might be called an "insertion file". If this condition exists, the user should be sufficiently familiar with EDIT so that he will be able to create and use this special temporary file.

Modifying a Section in a File

By now the reader should be reasonably familiar with the commands APPEND, WRITE, END, INSERT, SAVE AND GET.

The most common use of EDIT is to modify the contents of a file at a given point (typically, to correct an error). To make such a modification, the user must first read that section of the file into the buffer. Normally, a copy of the initial portion of the file is necessary, up to the APPEND which brings into the buffer the section to be modified. Now, the remaining EDIT commands are available to effect the modification. After the change is made, the process is terminated with an END command if modifying an existing file, or the Y command if the file is being created.

Some Command Examples

Below are several examples of useful command sequences to further acquaint the reader with EDIT's data manipulation facilities. In each example a command string is given and followed by a short explanation of what it will do.

(1) Assume the pointer is arbitrarily positioned within a line in the buffer:

0LT	Types the entire line leaving the pointer at its beginning.
0TT	Also types the entire line, but leaves the pointer unchanged.
0K	Erases the portion of the line to the left of the pointer.
K	Erases the portion of the line to the right of the pointer.
0LK	Erases the entire line.

For each of the following command sequences, it is assumed that n is sufficiently large.

BnK	Erases the entire buffer.
0X	Erases the entire SAVE area.
BnT	Prints the entire buffer.

(2) Assuming the pointer is positioned at the beginning of a line in the buffer,

nXnKZ-mLG

will move the next n lines to m lines from the end of the buffer and erase them from their original position.

(3) The command

Bn<mCI $L>,

for n sufficiently large, inserts a field of spaces in all lines at a point m characters from the beginning of each line.

(4) One can also scan the entire buffer with a FIND or SUBSTITUTE command by similarly using a sufficiently large numeric argument (called n below). The command will terminate when the end of the buffer is found with a CAN'T FIND message. For example:

Bn<Sfield1$field2$> will **replace all** occurrences of field1 by field2.

Bn<Ftext$-mD> will **delete all** occurrences of text, if m=the length of the text field.

Bn<Ftext$0LT1L> will **print** all lines containing text.

Bn<Ftext$0LK> will **delete** all lines containing text.

Bn<F;$I(CR)$> will **break** all lines containing semicolons into as many lines as there are semicolons - each terminating in a semicolon. (Note: In this case, any line originally ending in a semicolon will be followed by a "line" containing zero characters).

Bn<S$(CONTROL I)$L> will **replace** the first space in every line in the buffer by a horizontal tab control character.

Bn<A50T50K> will perform the following n times; append in the next (first) section, type it, and delete it from the buffer. This command string can be used to type a long file that can't be held all at once in the buffer. It is particularly useful in typing the listing output file of the assembler.

File Manipulation Summary

This section summarizes the steps needed to create a new file or to change an existing file.

Creating a New File

1. Use O (Output) to define the file that will be created. (Will default to drive 1 if drive is not specified).
2. Use I (Insert) to input text to buffer. End insert mode with ESCape ESCape ($$).
3. Use B,Z,C,L,D,K,X,G,F,S,T,P,*, or # as needed to edit.
4. Use Y (Close file) to output buffer contents to disk and to end the edit session.

Changing an Existing File

1. Use R (Read) to define the file that will be edited. (Will default to drive 0 if drive is not specified).
2. Use O (Output) to define the file that will be created. (Will default to drive 1 if drive is not specified).

3. Use A or N (Append) to bring lines from the input file into the editor buffer.
4. Use B,Z,C,L,D,K,X,G,F,S,T,P,*, or # as needed to edit.
5. If the entire file to be edited is too large to fit in the editor buffer, use W (Write) to write out edited text to the disk. Then repeat steps 3, 4, and 5 as needed.
6. Use E (End) to output buffer contents and/or the rest of the file to disk and end the edit session.

6. Disk Assembler (ASM8)

The computer understands only programs written in **machine** code, a sequence of hexadecimal characters. Most people, however, find that writing programs in machine code is usually tedious and often frustrating because of the need to keep track of where each instruction is located in memory and where all the variables are stored. An assembler is a program which automatically performs these housekeeping functions, allowing the user to write programs using convenient symbols, names, and expressions. The user can also add comments to his program to aid in debugging, and to make understanding and documenting easier.

The MS2000 disk assembler (ASM8) is such an assembler. It allows the user to program in assembly language. The ASM8 produces the machines code (hexadecimal) which can then be executed on the CDP1800-series microprocessors. A simple comparison of the same program in machine and assembly language, shown in Fig. 9, illustrates the ease of using assembly language. The ASM8 is designed to run under MicroDOS without the need of another computer. It includes level I, level II, macro, and cross-reference capability. Each of these capabilities is discussed in this chapter.

MACHINE CODE	ASSEMBLY LANGUAGE (ASM8)
F800 B8F8	ANSWER_AD EQU R8 ..OUTPUT ADDRESS WILL BE STORED HERE
24AB F80A	FIRST_NUM EQU 10
73F8 14F4	SECOND_NU EQU 20 ..THE TWO NUMBERS TO BE ADDED
5872 0000	A.1(ANSWER)→>ANSWER_AD.1;A.0(ANSWER)→>ANSWER AD.0 ..THESE COMMENTS ARE ALLOWED IN THE PROGRAM
	(FIRST_NUM+SECOND_NU)→>@ANSWER_AD ..ADD THE TWO NUMBERS
	ANSWER DS 1 ..AND STORE AT ANSWER

Fig. 9 - Machine code and ASM8 assembly language compared.

The assembly language program consists of a sequence of lines called the **source code**. Most of these lines are directly translated by ASM8 into machine code and placed in an output fill called the listing along with an echo (reprinting) of the source code. The hexadecimal portion of the listing is called the **object code** and is the machine-executable program. Some lines do not directly produce code, but rather tell the assembler to do something. These lines are called **directives**.

In this manual, the assembly language is described using illustrative examples and BNF notation. A full description of the language in BNF is given in Appendix B. BNF is a concise and easy-to-understand format for learning and reviewing assembly language.

Note: The MicroDisk Development System MS2000 can assemble and edit Microprocessor CDP1804, CDP1805, and CDP1806 instructions, and a hexadecimal or listing file that contains these instructions can be downloaded into the system under test through the Micromonitor CDP18S030 or through the MicroEmulator MSE3001. The CDP1804, CDP1805 and CDP1806 instructions can be run and debugged by the MicroEmulator but not by the Micromonitor. An alternative method of transporting assembled CDP1804, CDP1805, or CDP1806 code is to program it on a PROM and install the PROM into the system under test.

Assembler Operation

ASM8 is a two-pass assembler. In the first pass the symbol table consisting of user-defined labels and constants is created. In the second pass the object code and the listing are generated.

As AM8 runs, it simulates filling a memory with the machine-code equivalent of the user's source program. A two-byte location counter is used to point to the area in this simulated memory where the next piece of code is to be inserted. As each statement is coded, the hexadecimal equivalent is inserted in the actual object file on disk, and the location counter is advanced by the number of byes whose insertion into memory it has simulated. The programmer can also control and reference the location counter if he wishes. This simulation allows ASM8 to predict the results and effects of actual loading.

The most useful function of an assembler is keeping track of where branch points are and where variables are stored. To perform this function, an assembler builds a symbol table. Each identifier (defined later) is

entered in the table along with the address in memory that it stands for or whatever information is appropriate to it. The user references the symbol table whenever he uses an identifier. The user can add to the symbol table by defining an identifier. Both of these uses of the symbol table are described in greater detail later.

The user may often wish to use a numeric or literal constant in his program. He may wish to address two consecutive bytes in memory, for example. If he were programming in machine code, he would have to address each one of these bytes separately. The assembler evaluates simple expressions and allows the programmer to name one byte "WEIGHT," for example, and the next byte would then be "WEIGHT + 1." The use of this feature is explained in detail later.

Backus-Naur Format (BNF)

BNF notation is a concise and convenient way to express the syntax of a language. There are two major elements in notation: terminal and non-terminal elements. A terminal element is written exactly as it would appear when used; a non-terminal element is a description of something and always appears between angle brackets. For example:

<FIRST THREE LETTERS OF THE ALPHABET> ::= ABC

ABC is not a description of the item, it is the item itself. There are no commas between the letters, because a comma is not part of the alphabet. Likewise, there are no spaces between the letters as spaces are not part of the alphabet. "FIRST THREE LETTERS OF THE ALPHABET" is a description and appears between angle brackets. The symbol ::= can be read as "is defined as" and will be used in every definition. Where there is a choice between alternatives, the symbol ! will be used to separate the choice.

Examples:

<one> ::= 1
<plus sign> ::= +
<minus sign> ::= -
<tree> ::= <woody plant>
<binary digit> ::= 0!1

A binary digit could be either a 0 or a 1, but not both. A binary digit can be only a 0 or a 1. A decimal digit can be defined in two ways.

<decimal digit> ::= 0!1!2!3!4!5!6!7!8!9!
<decimal digit> ::= <binary digit>!2!3!4!5!6!7!8!9!

Notice that the decimal digit could be defined by explicity listing every possibility or by defining it in terms of already defined objects. The use of the description of a binary digit eliminates the need to explicity list 0 and 1.

Example:

<primary color> ::= <red>!<green>!<blue>
<American coin names> ::= PENNY!NICKEL! DIME!HALF-DOLLAR

Note that PENNY is the name itself and so is a terminal element. Red as a non-terminal element describes the color, not the name of the color.

<certain breed of dog> ::= <collie>!<German shepherd>!<beagle>
<certain name of dog> ::= REX!SPOT!SHAD! ROVER

If it were necessary to list every possible combination explicity, BNF would be an extremely voluminous description of anything. Fortunately, it is possible to describe an item recursively, using its own description as part of the description. An unsigned binary number can be defined recursively as follows:

<unsigned binary number> ::= <binary digit> !<binary digit> <unsigned binary number>

Under this definition 01 is an unsigned binary number because it is 0 (a binary digit) followed by 1 (an unsigned binary number) and 1 is an unsigned binary number because it is 1 (a binary digit). Both the first and second part of the definition were used. 03 is not an unsigned binary number because 03 is not a binary digit (0 or 1 only), and though 0 is a binary digit, it is not followed by an unsigned binary number. Because 03 does not satisfy either of the alternatives of the definition, it is not an unsinged binary number. Notice that under the definition of an unsigned binary number, any string of 1's and 0's of any length is an unsigned binary number. In practice, the computer has finite capacity and there are usually additional restrictions. These restrictions will be given as notes in the text.

Examples:

<forest> ::= <tree>!<tree><forest>
<crowd> ::=<person>!<person><crowd>

In reading BNF notation, blanks are ignored. Where a blank is required by syntax of the language, the special character Δ is used. In order to improve the readability of the BNF in the text, many of the spaces have been removed. If there is a question concerning syntax, the syntax description in Appendix B is complete and should be referred to.

It is important to remember that the assembler will be interpreting the program instructions using the syntax described in this manual.

Basic Definitions

Character Set

ASM8's character set includes all twenty-six uppercase letters, all ten decimal digits, and all other printing ASCII characters.

Character Strings, Identifiers, and Labels

A **character** is any of the characters in ASM8's character set. A character string is any sequence of characters. Any valid line of assembly language is a character string, but not any character string is a valid line of assembly code. An **identifier** is any character string of up to nine alphanumeric characters, beginning with a letter. An identifier may contain as many break characters as desired, but may not contain any special characters including spaces. If break characters are in any identifier they are counted as part of the nine alphanumeric characters that make up the maximum length identifier. A **label** is an identifier that is used to mark a location in the program. A label always begins in column 1, and ASM8 assumes that any identifier beginning in column 1 is a label.

<character string> ::= <character>!<character string><character>
<break character> ::=
<alphanumeric character> ::=<letter>! <decimal digit>!<break character>
<identifier> ::= <letter>!<identifier><alphanumberic character>
<label> ::= <identifier>
<space> ::= Δ! <space>Δ

Examples:

DFJSHRJQGQH	Character string (too many characters for an identifier)
FIRST_NUM	Character string and identifier
F	Character, character string, and identifier
1	Character and character string

Note that while an identifier is always a character string, a character string may not always be an identifier.

Constants

ASM8 recognizes two types of constants: numeric and literal. A **literal constant** is simply any character string between quotes. A common error is to forget the closing quote on a literal constant. The assembler then considers the rest of the line to be part of the literal constant.

<literal constant> ::= '<character string>'

When no other constants are defined on the same line, a literal constant can be 72 characters long.

There are four types of **numeric constants**: binary, octal, decimal, and hexadecimal. A binary constant is a string of 1's and 0's followed immediately by a B. An octal constant is a string of octal digits (0-7) followed immediately by a Q. A decimal constant is a string of decimal digits (0-9) followed immediately by a D. A hexadecimal constant is a string of hexadecimal digits (0-9, A, B, C, D, E, and F) followed immediately by an H. The D at the end of a decimal digit is optional. When ASM8 encounters a string of digits without either a B, Q, D, or H following it, it assumes that the string is a decimal constant. ASM8 immediately converts numeric constants to their hexadecimal equivalents and literal constants to their ASCII equivalents. All numeric constants are truncated to two bytes.

<binary digit> ::= 0!1
<octal digit> ::= <binary digit> !2!3!4!5!6!7
<decimal digit> ::= <octal digit> !8!9
<hexadecimal digit> ::= <decimal digit> !A!B!C!D!E!F
<binary constant> ::= <binary digit>B!<binary digit><binardy constant>
<octal constant> ::= <octal constant>Q!<octal digit><octal constant>
<decimal constant> ::= <decimal digit>!<decimal digit>D!<decimal digit><decimal constant>
<hexadecimal string> ::= <decimal digit>!<hexadecimal string><hexadecimal digit>
<hexadecimal constant>::=<hexadecimal string>H

Note that not spaces are allowed within numeric constants and that spaces within literal constants are considered valid parts of the constants.

Examples:

1	Decimal constant
1B	Binary constant
1Q	Octal constant
1H	Hexadecimal constant
1D	Decimal constant
1FH	Hexadecimal constant (equivalent to 31 decimal)
0F1H	Hexadecimal constant; note that because the first digit of any numeric must be a decimal digit, a leading zero is necessary here.
0000000000F1H	Hexadecimal constant; note that because of its length this constant is truncated to 00F1H.
93898838D	Decimal constant (equivalent to E3003H); note that because of its length, this constant is truncated to 3003H.
'9389838'	Literal constant; note that the quotes turn a decimal constant into

a ASCII-encoded literal constant. Quotes within literal constants are coded as ' ' (two quotes).

Errors:

F1H Interpreted as an identifier because it begins with a leter.

Keywords

ASM8 reserves several words for special use. These reserved words should not be used as identifiers because they may cause confusion if used in Level II statements. The mnemonics for the instruction sets of the CDP1800-series microprocessors are reserved keywords, as are the register names R0, R1, R2, etc. Other keywords will be mentioned throughout this manual. If a keyword is used as an identifier, ASM8 attempts to code it properly; but if unable to, ASM8 returns a duplicate-label error message.

Level I Assembly Language

Line and Statements

Obviously, not all combinations of characters result in valid lines of assembly language just as not all combinations of characters result in valid English sentences. An English sentence is made up of words and, in the same manner, a line of assembly is made up of statements.

There are four kinds of lines: executable, major, macro call, and minor. Each of these types of lines has a unique syntax. In machine code, there may be no spaces; but in ASM8, spaces may be added anywhere to improve readability. Normally, a space is a string of any number of blanks or spaces. A statement set is a string of up to ten executable statements (which will be defined later) separated by semicolons (;). Spaces may be arbitrarily inserted between executable statements in a statement set. A **comment** is any character string preceded by two periods (..) and may be added to any line to facilitate reading. ASM8 prints out the comment on the listing, but otherwise ignores it. Executable lines are lines that contain a major statement, and minor lines contain a minor statement. Executable lines may begin with a label in column 1. Anything other than an identifier must not begin in column 1. One can always add a label to any line that does not already have one, but except for use with executable lines, the labels are useless. Executable, macro call, major, and minor statements are discussed in the following pages. Each line ends with a carriage return and cannot be more than 80 characters long, exclusive of the carriage return.

<space> ::= Δ!<space>Δ

<statement set> ::= <executable statement>!<statement set> ; <statement set>

A statement set may not contain more than ten executable statements.

<comments> ::= ..<character string>

All lines must end with a carriage return and may or may not be commented.

<line ending> ::= <carriage return>!<comment> <carriage return>

<executable line> ::= <label> <statement set> <line ending>!<space><statement set><line ending>

<macro call line> ::= <label> <macro call statement><line ending>!<space><macro call statement><line ending>

<major line> ::= <label><major statement><line ending>!<space><major statement><line ending>

Labels with major lines are virtually useless, but are acceptable.

<minor lines> ::= <minor statement><line ending>

Expression Evaluation

A convenient feature of ASM8 is its ability to evaluate expressions in the source code. These expressions can then be used as the operands in various statements.

Arithmetic Expressions: As explained earlier, ASM8 keeps a location counter that points to the address in the simulated memory where the next piece of machine code is to be placed. The value of this location counter can be used in an expression by using the symbol, $. Likewise, the value of an identifier, once defined, may be used in an expression by merely using its name. A term (explained below) may be used by putting it in parentheses according to normal algebra practice. A constant can be used in an expression, but whenever a constant is used, only the last two bytes of its hexadecimal equivalent are used. When evaluating an expression, ASM8 normally carried two bytes, but often the programmer will wish to address only the upper or lower byte of a number. The programmer can do so by using the operators A.0(*) to extract the low-order byte of *, or by using A.1(*) to extract the high-order byte of *. (* is used here to represent a term, which will be explained later.) No spaces may appear between the period and either the A or binary digit. An expression may also contain special elements called dummies. Dummies are identifiers within brackets,[ ], and always stand for another identifier or constant. Their use is explained later. The location counter, a constant, a literal constant, an identifier, the least or most significant byte, and a dummy are all known as arithmetic elements. If a literal constant is used, it is truncated to its last two bytes.

<location counter> ::= $
<dummary> ::= [<identifier>]
<least significant byte> ::= A.0(<term>)
<most significant byte> ::= A.1(<term>)
<element> ::= <identifier>!<constant>!<location counter>!<dummy>!<least significant bytes>!<most significant byte>!<term>)

Examples:

$	Location counter
[FIVE]	Dummy
A.0(ADDRESS)	Least significant byte of address
015H	Constant
A.1(ADDRESS)	Most significant byte of address
TIME	Identifier
($ * 2 + 4)	(Term)
'A'	Literal constant (equivalent to 0041H)

Errors

4 + 3	This term is not in parentheses

Expressions can be built up according to the normal rules of algebra. Factors may be multiplied together or divided to produce other factors. Terms can be added together or subtracted. Except where parentheses override the hierarchy, negation is performed first followed by multiplication or division from left to right, and then by addition or subtraction from left to right.

<factor> ::= <element>!+<element>!-<element>!<factor>*<factor>!<factor>/<factor>
<term> ::= <factor>!<term>+<term>!<term>-<term>

Examples:

A+B	Term
A*B	Factor, term
A.0(ADD) + 5	Term
(A+B)	Element, factor, term
(5+3)*2-6	Term (evaluates to 10)

Relational Expressions: The term is the highest form of arithmetic result. But, because for certain statements logical results are needed, ASM8 is capable of comparing two terms to obtain a logical result. There are six relational operators, .EQ., .GT., .LT., .LE., .GE., and .NE.. The result of a comparison can be "NOTTED" by use of the operator .NOT.. Spaces may be inserted arbitrarily before or after any relational operator.

<relational operator> ::= .EQ.!.LT.!.GT.!.LE.!.GE.!.NE.
<relation> ::= <term> <relational operator> <term>!.NOT. <relation>

Examples:

5.EQ.5	The result is true
.NOT.5.EQ.5	The result is false
3.GT.5	False
3 .GE. 5	False
3.LT.5	True
5.NE.5	False

Errors:

3. LT. 5	The . must immediately follow and precede the letters in the relational operator. This example is read as a constant followed immediately by a space and character string.

Logical Expressions: Just as arithmetic expressions can be built by the rules of ordinary algebra, logical expressions can be built by the rules of Boolean algebra. The three operators are .AND., .XOR., and .OR.. The result of an .AND. operation is true if and only if both operands are true. The result of an .XOR. operation is false if the two operands are equal and true if the operands are unequal. The result of an .OR. operation is true if either or both of the operands is true. .AND. operations are performed first, followed by .XOR. and .OR. operations, except where parentheses are used to override the hierarchy. Spaces may be inserted arbitrarily before or after the operators.

<logical element> ::= <relation>!(<logical term>)!<logical element> .AND. <logical element>
<logical factor> ::= <logical element>!<logical factor> .XOR. <logical factor>
<logical term> ::= <logical factor>!<logical term> .OR. <logical term>

Examples:

ADR .GT. 1000H .AND. A.0(ADR) .EQ. 0	Logical element
.NOT. (ADD .LT. FIVE+BEGIN) .OR. THIS .GT. THAT	Logical term
5*TEN - -6 .EQ. 0 .AND. B.EQ.EIGHT .OR. A .EQ.B	Logical term
'THIS' .EQ. 'THAT' .OR. 'I' .EQ. 'I'	Logical term

Errors:

NOT FIVE AND ONE	Without the periods, this example is interpreted as four identifiers.

Bitslice Expressions: When ASM8 encounters a relation, it evaluates one in the same way that it evaluates an arithmetic operation, except that it returns only one of two values: OFFFFH (-1H) for true, and 0000H (0) for false. The logical operators actually work on a bit-by-bit basis so that a term may be used as a logical element instead of a relation. Because this facility can lead to programming complications, it is not recommended that the beginning programmer use it.

Examples:

0101B .AND. 0011B	Equivalent to 0001B

0101B .XOR. 0011B	Equivalent to 0110B
0101B .OR. 0011B	Equivalent to 0111B
.NOT. 0101B	Equivalent to 1010B

Limitation: Because the assembler must store partial results to expressions, there are limits to the size and complexity of expressions that can be evaluated. The general guideline is never to use an expression that has more than twenty elements or twenty operators. An operator is any of the normal logical, relational, or arithmetic operators.

<arithmetic operator> ::= +!-!*!/
<byte extraction operator> ::= A.0(!A.1)
<relational operator> ::=
.EQ.!.NE.!.LT.!.GE.!.LE.!.GT.
<logical operator> ::= .NOT.!.AND.!.OR.!.XOR.

Executable Statements: Level I

Level I executable statements consist of CDP1800-series mnemonics and the appropriate operands. The CDP1800-series instruction set can be divided into four classes. The first class contains those instructions that have no operands. The second class of instructions includes those that require a single operand which must be a register. The third class includes those that require an immediate operand. The fourth class contains those instructions that require both a register and an immediate operand.

A **register** is any hexadecimal constant, an R followed immediately by a hexadecimal digit, or a term. Only the last four bits of the hexadecimal digit or term result are used. Some of the third class instructions require operands that are only one byte. If the operand given or evaluated is longer than one byte, the low-order byte is used. If the instruction requires two bytes and the operand given or evaluated is only one byte, the high-order byte is 0. an operand string is a set of immediate operands and registers, separated by commas. There can be no more than 49 characteres in the operand string.

In summary, the operand must be appropriate to the instruction. An executable statement is any first class instruction, a second class instruction and a register, a third class instruction and an immediate operand, or a fourth class instruction, a register, and an immediate operand.

<register> ::= <term>!R<hexadecimal digit>
<immediate operand> ::= <term>
<operand string> ::= <immediate operand>
!<register>
!<operand string>, <operand string>

First Class Instructions:
For all types: IDL, NOP, SEQ, REQ, SAV, MARK, RET, DIS, LDX, LDXA, STXD, IRX, OR, XOR, AND, SHR, SHRC, SHL, SHLC, ADD, ADC, SD, SDB, SM, SMB, SKP, LSKP, LSZ, LSNZ, LSNF, LSQ, LSNQ, LSIE

For types CDP1805C, CDP1806C, CDP1804AC, CDP1805AC and CDP1806AC only: LDC, GEC, STPC, DTC, STM, SCM1, SCM2, SPM1, SPM2, ETQ, XIE, XID, CIE, CID, BCI, BXI

For types CDP1804AC, CDP1805AC, and CDP 1806AC only: DADD, DADC, DSM, DSMB, DSAV.

Second Class Instructions:
For all types: SEP, SEX, LDN, LDA, STR, INC, DEC, GLO, PLO, GHI, PHI

For types CDP1805C, CDP1806C, CDP1804AC, CDP1805AC, and CDP1806AC only: RLXA, RSXD, RNX, SRET.

Third Class Instructions:
For all types: LDI, ORI, XRI, ANI, ADI, ADCI, SDI, SDBI, SMI, SMBI, BR, NBR, BZ, NBZ, BDF, BPZ, BGE, GNF, LBR, LBZ, LBNZ, LBDF, LBQ, LBNQ, NLBR, BM, BL, BQ, BNQ, OUT, INP

For types CDP1804AC, CDP1805AC, and CDP 1806AC only: DADI, DACI, DSMI, DSBI.

Fourth Class Instructions:
For types CDP1805C, CDP1806C, CDP1804AC, CDP1805AC, and CDP1806AC only: RLDI, SCAL

For types CDP1804AC, CDP1805AC, and CDP 1806AC only: DBNZ

<executable statement> ::= <first class instruction>
!<second class instruction><register>
!<third class instruction><immediate operand>
!<fourth class instruction><register>,
<immediate operand>

Examples:

LDI FIVE + FOUR	Third class
LDX	First class
CALL UCALL, TYPE, BUFFER	Fourth class (CALL is explained later
STR RF	Second class

Errors:

LDI	LDI requires an operand; it is thrid class
L DI	No spaces are allowed in instruction mnemonics

Macro Call Statement. A macro is explained in detail later, but it can be thought of as a user-defined mnemonic. Once defined, it can be used in the same manner as any other mnemonic except that it may not be part of a statement set. A macro call statement consists of the

macro name followed by a space and an operand string if appropriate. The operands that make up the operand string must be in the order and type that is correct for that macro. Because the assembler cannot know what the programmer's macro does, it cannot tell if it has been provided with an incorrect operand string. The macro name can be any identifier.

<macro name> ::= <identifier>
<macro call statement> ::= <macro name>
<operand string>

Directives. As stated earlier, certain lines of the source code do not directly result in a piece of machine mode. These directives use keywords similar to mneomonics called **pseudo-ops**. There are two types of directives, the major and minor statements. The minor statements are used to change the location counter or the symbol table. The minor statements must begin in column 1. Two of them must begin with a label, and three must begin with either a label or a space in column 1. None of the major statements may begin in column 1, but like the executable statements, all may have an operational label preceding them.

Minor Statement. There are five types of minor statements. The first of these statements, the simplest, is used to change the symbol table. It is called the **EQUATE** statement. The EQUATE statement consists of a label (beginning in column 1) followed by a space, the word EQU, another space, and an immediate operand, a label, or a register. When ASM8 encounters an EQUATE statement, it puts the label in the symbol table along with the value that it is equated to.

The second type of minor statement is the **constant declaration.** It consists of an optional label followed by a space, the word DC, another space, and an operand string. When the assembler encounters a constant declaration it simply places the immediate operands directly into the object code, with the exception that literal constants are not truncated to two bytes.

The third type of minor statement, is the **storage declaration.** It is an optional label followed by a space, the word DS, another space, and a term. When the assembler encounters a storage declaration it defines the label as the starting address of a buffer area whose length is equal to the term. In handling both the constant and storage declarations, ASM8 advances the location counter by the number of bytes inserted. Two statements, the ORG and PAGE statements change the location counter directly. The ORG statement consists of an optional label followed by a space, the word ORG, another space, and a term. The location counter is set equal to the value of the term. The PAGE statement consists of an optional label followed by the word PAGE, and it sets the location counter to the start of the next page. (A page is equal to 256 bytes.)

<equate statement> ::= <label> EQU <term>
!<label> EQU <register>
<constant declaration> ::= <label> DC
<operand string>
!<space> DC <operand string>
<storage declaration> ::= <label> DS <term>
<org statement> ::= <label> ORG <term>
!<space> ORG <term>
<page statement> ::= <label> PAGE
!<space> PAGE

Examples:

FIVE EQU 5
Equate statement
OUTPUT DS 10
Storage declaration (10 bytes)
OUTPUT ORG $+10
Advance the location counter by 10 bytes and label the first byte. Note that this statement is equivalent to the statement above
DC 'THE QUESTION'
Constant declaration (ASCII encoded)
INPUT DS INP_LENGTH
Storage declaration
DC 568393H, 5798192H
Constant declaration (truncated to 83938192H)
NEWPAGE PAGE
Page statement

Sample Program - Level I. Fig. 10 is a sample program that illustrates some of the elements of level 1 assembly language that have already been covered.

Major Statements. There are two types of major statements: status and conditional assembly.

Status Statements. The status statements are the simpler of the two sets. There are six types of status statements. The simplest is the END statement which tells ASM8 that there are no more assembly lines to process and to ignore anything that follows. This statement should be the last line of any program. The next statement, the EJECT statement, tells ASM8 to insert a top-of-form character in the output. It does not affect the processing. A NOLIST statement directs the assembler to cease echoing the source code to the listing. The machine code is still inserted in the listing, but the source code is no longer printed. A LIST statement tells the assembler to resume echoing the source code and thus cancels the effect of the NOLIST statement. Each of these statements consists of a keyword that may be arbitrarily preceded or succeeded by spaces. The keywords are END, EJECT, NOLIST, and LIST. The remaining two major statements are used with macros. They are used to indicate the beginning and end of a macro. These

```
    ..THIS PROGRAM IS A SAMPLE PROGRAM.
    ..IT WILL ADD TWO NUMBERS TOGETHER.
    ..THIS PROGRAM IS NOT EFFICIENT, BUT IS INTENDED TO
    ..ILLUSTRATE THE USE OF ASSEMBLY LANGUAGE.
FIRST_NUM EQU 25              ..THE NUMBERS ARE DEFINED SO THAT THEY
SCND_NUM EQU 31               ..CAN BE CHANGED EASILY
UTILITY EQU R8                ..REGISTER 8 WILL BE USED AS A TEMPORARY
    LDI FIRST_NUM             ..PUT THE FIRST NUMBER IN THE D REGISTER
    PLO UTILITY               ..PUT IT IN THE LO ORDER BYTE OF THE
                              ..TEMPORARY
    ANI 0                     ..CLEAR THE D REGISTER
    PHI UTILITY               ..CLEAR HI ORDER BYTE OF THE TEMPORARY
    LBR ADD_NUMS              ..BRANCH AROUND THE NEXT AREA
  DC  OF8CCH, 134DH           ..ADD A CONSTANT FOR NO REASON
ADD_NUMS GLO UTILITY          ..PUT THE LO ORDER BYTE OF THE TEMPORARY
                              ..INTO THE D REGISTER
    ADI SCND_NUM              ..ADD THE SECOND NUMBER
    PLO UTILITY               ..PUT THE SUM BACK IN THE TEMPORARY
    LDI A.1(ANSWER);PHI R7    ..PUT THE HI ORDER BYTE OF THE ANSWER's
                              ..ADDRESS IN R7 FOR LATER USE
    LDI A.0(ANSWER);PLO R7    ..PUT THE REST OF THE ADDRESS IN R7
    GLO UTILITY               ..GET THE SUM
    STR R7                    ..AND PUT IT IN THE ANSWER BUFFER
    IDL                       ..STOP
ANSWER  DS  1                 ..SET ASIDE ONE BYTE FOR THE ANSWER
```

Fig. 10 - Sample Level I assembly language program.

statements are explained later in detail, but they are presented here because they have the same form and function as the other major statements.

<end statement> ::= END <label>!END
<eject statement> := EJECT
<nolist statement> ::= NOLIST
<list statement> ::= LIST
<macro statement> ::= MACRO
<endm statement> ::= ENDM

Remember that because major lines do not have labels, all of these statements must begin in a column other than 1.

Examples:

END	END statement
EJECT	EJECT statement
NOLIST	NOLIST statement
LIST	LIST statement
MACRO	MACRO statement
ENDM	ENDM statement

Conditional Assembly Statements. Conditional assembly statements tell the assembler to assemble portions of the source code only if certain conditions are met. A **LINE** block is a sequence of lines. An **IF** block begins with an IF line and ends with an ENDIF line. There is an ELSE line between the IF and the ENDIF lines. When ASM8 encounters an IF line, it evaluates the logical term. If the result is true, then the statements between the IF line and the ELSE line are processed. If the result is false, then the statements between the ELSE line and the ENDIF line are processed. The IF line consists of the keyword IF followed by a logical term separated by a space. The ELSE and ENDIF statements have the same format as the status statements, using the keywords ELSE and ENDIF.

The IF blocks can be nested (an IF block can contain an IF block) but it must be remembered that the assembler associates an ELSE or ENDIF line with the IF line that most recently preceded it. It is good practice to always include the ELSE statement explicity in the source code.

<if statement> ::= IF <logical term>
<else statement> ::= ELSE
<endif statement> ::= ENDIF
<line block> ::= <line>!<line block><line>
<if block> ::= <if statement><line block>
<else line><line block><endif line>

Remember that each line is separated from the next by a carriage return, and that the line blocks could be empty (contain no lines).

The next type of conditional assembly block is the **DO** block. The DO block consists of a DO line, followed by a LINE block and then by an ENDD line. The DO statement consists of the keyword DO, a space, a

dummy, and then either an = and an increment list, or a : and a list of replacement values. The increment list consists of three expressions separated by commas. Each of these expressions is truncated to 1 byte, so that its range is from 0 to 255. The replacement list consists of a series of terms separated by commas. The values of the terms in a DO line may not be changed within the DO block. An attempt to do so will result in incorrect code.

If the = and increment list are used, then the lines within the DO block are assembled several times. The first time they are assembled, the dummy has the value of the first constant in the increment list, called the beginning value. The third constant is called the step value, and the dummy is incremented by the step value each time the DO block is assembled. The second constant is called the ending value. The assembler continually increments the dummy until its value exceeds the ending value. It then resumes normal processing after the ENDD statement. If the : and replacement list are used, then the dummy takes on a different value from the replacement list each time the block is assembled until there are no more values left in the list.

A DO block may be nested within another DO

```
..THIS IS A SAMPLE OF WHAT CAN BE DONE WITH MAJOR STATEMENTS
ONE EQU 1
TWO EQU 2
  IF ONE .EQ. TWO                  ..IS THIS TRUE?
    LDI ONE                        ..IS SO THEN LOAD ONE IMMEDIATE
    ELSE
    IF TWO .EQ. ONE                ..IF NOT TRY AGAIN
    ELSE                           ..THERE IS NO TRUE PART
      DO [I] = 1,2,1 LDI [I]       ..DO THIS TWICE
      ENDD
    ENDIF
  ENDIF
  GO FORWARD
  THIS IS JUNK WHICH WILL BE IGNORED
FORWARD  PAGE                      ..ADVANCE TO THE NEXT PAGE
         ORG 1111H                 ..CHANGE THE LOCATION COUNTER
  DO[I]:ONE,TWO,ONE,TWO
    LDI [I]
  ENDD
  NOLIST                           ..STOP ECHOING THE SOURCE
                                   ..IT WILL NOT PRINT THIS COMMENT
  LIST
  END
```

Fig. 11(a) - Sample program illustrating major statements source code.

```
!M
0000 ;          0000    ..THIS IS A SAMPLE OF WHAT CAN BE DONE
                        ..WITH MAJOR STATEMENTS
0000 ;          0002    ONE EQU 1
0000 ;          0003    TWO EQU 2
0000 F801 ;     0009            LDI 1       ..DO THIS TWICE
0002 F802 ;     0009            LDI 2       ..DO THIS TWICE
0004 ;          0016    FORWARD PAGE        ..ADVANCE TO THE NEXT PAGE
0100 ;          0017            ORG   1111H       ..CHANGE THE LOCATION
                                              ..COUNTER
1111 F801;      0018            LDI   ONE
1113 F802;      0018            LDI   TWO
1115 F801;      0018            LDI   ONE
1117 F802;      0018            LDI   TWO
1119 ;
1119 ;
0000
```

Fig. 11(b) - Sample program illustrating major statements listing.

block, but the assembler associates an ENDD statement with the DO line that most recently precedes it.

<beginning value> ::= <constant>
<ending value> ::= <constant>
<step value> ::= <constant>
<do statement> ::= DO <dummy> :
<operand string>
!DO <dummy> = <beginning value> ,
<ending value> , <step value>
<endd statement> ::= ENDD
<do block> ::= <do line><line block>
<endd line>

Remember that each line is separated by a carriage return.

There is one remaining conditional assembly statement - the GO statement. The format for the **GO** statement is GO followed by a space and a label. When the assembler encounters a GO statement, it stops processing the source code until it finds the label. Because the assembler cannot find the label if it precedes the GO statement, it must not precede.

<go statement> ::= GO <label>

When the conditional assembly statements are used, it should be remembered that a GO statement cannot point to a label that is outside the DO or IF block the go line is in, or to a label that precedes it.

Sample Program - Major Statements. The sample program in Fig. 11 illustrates the use of major statements. Immediately following the source code, Fig. 11(a), is the listing, Fig. 11(b). A comparison of the two illustrates how the major statement directs the assembler.

Level II Assembly Language

In order to make programming easier, in Level II operations several of the op-code mnemonics can be replaced with codes that correpond to their most frequent use. Likewise, operations involving the D register can be done using D-sequence instructions. In D-sequence instructions, special characters are used instead of op-code mnemonics making D-sequence instructions similar in appearance to APL statements. (APL is a high-level programming language).

Executable Statements: Level II

Substitution Instructions. The substitutions for the op-code mnemonics fall into two forms. The mnemonics and their substitutions are listed in Table VI. The first form involves simply the use of an immediate keyword in the same way that the mnemonic was used. These keywords are IDLE, GOTO, NOGOTO, SKIP, RETURN, DISABLE, POP, PUSH, SAVE, GOSTATE, CALL, and EXIT. EXIT is treated like a first class instruction and CALL is treated like a macro call in that it is followed by an operand string. They are used to execute the standard call and return procedures. In order to use them, the registers 2 through 6 must already be set aside for the standard call and return procedure. They can be initialized by using the Utility Program UT71 built-in subroutines, INIT1 and INIT2 (Refer to Chapter 8). The operands of CALL consist of the address of the subroutine, followed by any inline parameters that the programmer wishes to pass. EXIT has no operands.

The second form consists of the word IF followed by a space, a BRANCH keyword, another space, and the keyword GOTO. The BRANCH keywords indicate the condition on which a branch is to take place. They are =0, Q, &=0, DF, PZ, GE, EF1, EF2, EF3, EF4, NQ, &>0, >0, NDF, MINUS, LESS, NEF1, NEF2, NEF3, and EF4.

Table VI - Level II Substitutions for Level I Mnemonics

Level I	Level II
B1	IF EF1 GOTO
B2	IF EF2 GOTO
B3	IF EF3 GOTO
B4	IF EF4 GOTO
BDF	IF DF GOTO
BGE	IF GE GOTO
BL	IF LESS GOTO
BM	IF MINUS GOTO
BN1	IF NEF1 GOTO
BN2	IF NEF2 GOTO
BN3	IF NEF3 GOTO
BN4	IF NEF4 GOTO
BNF	IF NDF GOTO
BNQ	IF NQ GOTO
BNZ	IF &>0 GOTO IF >0 GOTO
BPZ	IF PZ GOTO
BR	GOTO
BQ	IF Q GOTO
BZ	IF &=0 GOTO IF =0 GOTO
DIS	DISABLE
IDL	IDLE
LDXA	POP
NBR	NOGOTO
RET	RETURN
SAV	SAVE
SEP	GOSTATE
SKP	SKIP
STXD	PUSH
SEP R4	CALL
SEP R5	EXIT

```
<immediate keyword> ::= IDLE!GOTO
   !NOGOTO!SKIP!RETURN!DISABLE!POP
   !PUSH!SAVE!GOSTATE!CALL!EXIT
<branch keyword> ::= O!Q!&=O!DF!PZ!GE!
   !EF1!EF2!EF3!EF4!NQ!&>O!>O!NDF!
   !MINUS!LESS!NEF1!NEF2!NEF3!NEF4
<substitution> ::= IF <branch keyword>
   GOTO!<immediate keyword>
```

Examples:

IDLE	IDL
GOTO ADD_NUMS	BR ADD_NUMS
IF =O GOTO BEGINNING	BZ BEGINNING
IF NEF4 GOTO END	BN4 END
GOSTATE R5	SEP R5
CALL TYPE, 'MESSAGE'	SEP R4; DC TYPE DC 'MESSAGE'
PUSH X	STXD X
POP Y	LDA Y

D-Sequence Instructions. The D-Sequence instructions consists of three parts; the load part, the manipulation part, and the storage part. What each of these parts corresponds to is listed in Table VII. Not all parts are needed in a statement. Any single part can be present or all can be present. Two parts can also be present, but if more than one part is present, the order load, manipulation, and storage part must be maintained.

The load part tells the assembler what should be loaded into the D-register. A register name followed by a .0 or .1 indicates that either the low- or high-order byte of that register should be loaded into D. A constant, identifier, or term in parentheses indicates that the value of that constant, identifies or term should be loaded immediately into the D-register. An @ indicates that the D-register should be loaded from memory. If a register name follows the @, then the byte pointed to by that register is used. If no register name is specified, the register named by the X register is used. If a "precedes the register name it indicates that the X-register should be set to point to that register. If memory is accessed and a ! ends the load part, the contents of the register used is incremented. If the @ ends the load part, a comment in parentheses may be inserted immediately (without spaces) after the @.

The manipulation part tells the assembler what is to be done with the D-register. There are 9 binary operations which can be performed and 4 unary operations. The binary operations are + (add), - (subtract), -+ (subtract and negate), +" (add with carry), -" (subtract with borrow), -+" (subtract and negate with borrow), .AND. (and), .OR. (or), and .XOR. (exclusive or). The manipulation part for the binary operations consists of the operator symbol followed without spaces by the source of the second operand. The source can be a memory location, a constant, an identifier, or a term in parentheses. If a constant, identifier, or term is used, its value is immediately used. To use the memory, an @ immediately follows the operation symbol. Immediately following the @ there is a " followed by a register name. The X-register is set to register name and the register points to the memory byte that is used. The unary operators are /2 (shift right), *2 (shift left), /2" (shift right circular) or *2" (shift left circular).

The storage part tells the assembler what to do with the contents of the D-register. All storage parts begin with -> (a minus followed by a greater than). If a register name followed by .0 or .1 follows the arrow (->), the contents are stored in the low- or high-order byte of that register. If an σ follows the arrow, the contents are stored in memory. If a register name follows the σ, it points to the byte in memory where the D-register contents are to be stored. If no register name follows the σ, the register specified by the X-register is used. The σ may be followed by a - indicating that the contents of the register used should be decremented. If the - is used, then the register name (if there is one) must be separated from the - by a ". The X-register is set to the register name given. If the σ- is the end of the storage part, then a comment within parentheses may immediately follow the σ-.

```
<load part> ::= @!@!!@<register>!
   !@<register>!
   !@"<register>!@( <character string>)
   !<register>.0!<register>.1!<term>
<object> ::= @!@"<register>!<term>
<operator> ::= +!-!-+!+"!-"!-+"!.AND.
   !.OR.!.XOR.
<manipulation part> ::= <operator><object>
   !/2!*2!/2"!*2"
<storage part> ::= -><register>.0
   !-><register>.1
   !->@<register>!->@-!->@-"<register>
   !->@-(<character string>)
<D-sequence statement> ::= <load part>
   !<manipulation part>!<storage part>
   !<load part><manipulation part>
   !<load part><storage part>
   !<manipulation part><storage part>
   !<load part><manipulation part>
      <storage part>
```

Note that no spaces are allowed between the special characters involved or between the special characters and any identifiers or registers that are used. There is also a limit on the length of a Level II statement. It may contain no more than thirty-nine characters.

Table VII - D-Sequence Statements

Symbol	Level I	Action
Load Part		
@	LDX	M(R(X))→>D
@"N	SEX N;LDX	N→>X;M(R(X))→>D
@(COMMENT)	LDX ..COMMENT	M(R(X))→>D
@N	LDN N	M(R(N))→>D FOR N<>0
N.0	GLO N	R(N).0→>D
N.1	GHI N	R(N).1→>D
@N!	LDA N	M(R(N))→>D;R(N)+1→>R(N)
CONSTANT	LDI CONSTANT	A.0(CONSTANT)→>D
@!	LDXA	M(R(X))→>D;R(X)+1→>R(X)
Manipulation Part		
+@	ADD	D+M(R(X))→>DF,D
+@"N	SEX N;ADD	N→>X;D+M(R(X))→>DF,D
+CONSTANT	ADI CONSTANT	D+CONSTANT→>DF,D
-@	SM	D-M(R(X))→>DF,D
-@"N	SEX N;SM	N→>X;D-M(R(X))→>DF,D
-CONSTANT	SMI CONSTANT	D-CONSTANT→>DF,D
-+@	SD	M(R(X))-D→>DF,D
-+@"N	SEX N;SD	N→>X;M(R(X))-D→>DF,D
-+CONSTANT	SDI CONSTANT	CONSTANT-D→>DF,D
+"@	ADC	D+M(R(X))+DF→>DF,D
+"@"N"	SEX N;ADC	N→>X;D+M(R(X))+DF→>DF,D
+"CONSTANT	ADCI CONSTANT	D+CONSTANT+DF→>DF,D
-"@	SMB	D-M(R(X))-NDF→>DF,D
-"@"N	SEX N;SMB	N→>X;D-M(R(X))-NDF→>DF,D
-"CONSTANT	SMBI CONSTANT	D-CONSTANT-NDF→>DF,D
-+"@	SDB	M(R(X))-D-NDF→>DF,D
-+"@"N	SEX N;SDB	N→>X;M(R(X))-NDF→>DF,D
-+"CONSTANT	SDBI CONSTANT	CONSTANT-D-NDF→>DF,D
.AND.@	AND	D.AND.M(R(X))→>D
.AND.@"N	SEX N;AND	N→>X;D.AND.M(R(X))→>D
.AND.CONSTANT	ANI CONSTANT	D.AND.CONSTANT→>D
.OR.@	OR	D.OR.M(R(X))→>D
.OR.@"N	SEX N;OR	N→>X;D.OR.M(R(X))→>D
.OR.CONSTANT	ORI CONSTANT	D.OR.CONSTANT→>D
.XOR.@	XOR	D.XOR.M(R(X))→>D
.XOR.@"N	SEX N;XOR	N→>X;D.XOR.M(R(X))→>D
.XOR.CONSTANT	XRI CONSTANT	D.XOR.CONSTANT→>D
/2	SHR	SHIFT D RIGHT NONCIRCULAR
*2	SHL	SHIFT D LEFT NONCIRCULAR
/2"	SHRC	SHIFT D RIGHT CIRCULAR
*2"	SHLC	SHIFT D LEFT CIRCULAR
Storage Part		
→>N.0	PLO N	D→>R(N).0
→>N.1	PHI N	D→>R(N).1
→>@N	STR N	D→>M(R(N))
→>@-	STXD	D→>M(R(X));R(X)-1→>R(X)
→>@-"N	SEX N;STXD	N→>X;D→>M(R(X));R(X)-1→>R(X)
→>@-(COMMENT)	STXD ..COMMENT	D→>M(R(X));R(X)-1→>R(X)

Note 1: Whereever an N appears, a register may be placed. (R followed by ahexadecimal digit or a hexadecimal constant less than 10H).

Note 2: Wherever the word constant appears, a constant or valid identifier may be placed.

Note 3: Wherever an @ appears at the end of a part (not followed by "N, N, or !), it may be replaced with @ (comment).

Note 4: Note that →>@ will result in STXD instruction.

Examples:

```
  5→>R5.0              LDI 5;PLO R5
  5                    LDI 5
  A                    LDI A
  FIVE+2→>R7.0         LDI FIVE;ADI 2;PLO R7
  @N!→>@-"N            LDA N;SEX N;STXD
  .XOR.CAR_RET         XRI CAR_RET
  (FIVE+SIX)→>
@UTILITY               LDI 11;STR UTILITY
```

Sample Program Illustrating D-Sequences. Fig. 12 is a repeat of Fig. 11, the first sample program written in Level II assembly. It illustrates the use of the D-sequence statements and substitutions.

Macros and Their Use

A **macro** is a programmer-defined collection of statements that, in its entirety, has been assigned a special mnemonic or name by the programmer. Once a macro has been defined, the programmer may call in the macro by the use of its name in the same way that a normal mnemonic would be used. When the assembler encounters a mnemonic that is not a normal op-code, mnemonic, or identifier, it checks to see if it is a macro name. If it is, the assembler substitutes the lines of the macro into the listing. This process is called text insertion or macro expansion.

When the assembler inserts the text of a macro into the listing it can make changes to the text in two basic ways. The calling line may have parameters in the form of operands which are to be substituted for certain dummies in the macro. Using the major directives for conditional assembly, the programmer may direct the assembler to assemble only portions of the macro text.

It is important that a programmer understand the difference between a macro and a subroutine. A **subroutine** is a subprogram which occupies a single memory area but can be called several times from various locations through a process called subroutine linkage. A macro is a set of lines of assembly language that are inserted at assembly time. The macro approach eliminates all linkage problems and is faster in execution, but probably results in more code than the subroutine approach.

A collection of macros in a single file is called a **macro library.** Effectively, a macro library extends the set of op-code mnemonics. The capabilities of the machine as seen by the assembly programmer can be greatly expanded by the use of a good macro library.

ASM8 recognizes macros in two locations. They may be in the same file as the main program (though not interspersed with it) or they may be in a special file containing a macro library.

The Mechanics of Macro Usage

In order to allow the programmer to use macros, three major statements have already been introduced. They are the MACRO, ENDM and EXITM statements. The MACRO statement instructs the assembler that the statements that follow are part of a macro and should be the first line of any macro. The ENDM statement tells the assembler that the end of the macro has been reached and should be the last line of any macro. The EXITM statement tells the assembler to cease processing statements until it encounters an ENDM statement.

The second line, immediately following the MACRO statement must be the macro definition. The macro definition consists of the name of the macro followed by a space and dummy list. The dummy list is a sequence of dummies separated by commas and may have an arbitrary number of spaces around the commas. At assembly time, these dummies are replaced throughout the macro by the corresponding operands of the calling statement.

```
    ..THIS IS A REPEAT OF THE PROGRAM
    ..TO ADD TWO NUMBERS TOGETHER.
FIRST_NUM EQU 25                ..THE NUMBERS ARE DEFINED SO THEY CAN
SCND_NUM EQU 31                 ..BE EASILY CHANGED
UTILITY EQU R8                  ..REGISTER 8 WILL BE USED AS A TEMPORARY
    FIRST_NUM→>UTILITY.0        ..PUT THE FIRST NUMBER INTO THE
                                ..LOW ORDER BYTE OF R8
    .AND.0→>UTILITY.1           ..CLEAR THE HIGH ORDER BYTE OF R8
    GOTO ADD_NUMS               ..USE A SUBSTITUTE
  DC    OF8CCH,134DH            ..MAJOR STATEMENTS ARE UNCHANGED
ADD_NUMS UTILITY.0+SCND_NUM→>
  UTILITY.0                     ..ADD THE SECOND NUMBER
    A.0(ANSWER)→>R7.0           ..PUT THE ANSWER'S ADDRESS IN R7
    A.1(ANSWER)→>R7.1
    UTILITY.0→>@R7              ..STORE THE ANSWER
    IDLE
ANSWER  DS  1
```

Fig. 12 - Sample program illustrating use of D-sequence.

<macro statement> ::= MACRO
<endm statement> ::= ENDM
<exitm statement>::= EXITM
<dummy list> ::= <dummy>
!<dummy list>,<dummy list>
<macro definition> ::= <macro name>
<dummy list>
<macro> ::= <macro line>
<macro definition line>
<line block><endm line>

Examples:

```
TYPE [MESS_LENG],[MESSAGE]
LOOK [REGISTER]
FIND [CHARACTER],[SUBSTITUTE], [END]
NEXT
TIME
```

In order to operate with dummies, the assembler must keep a substitution list. For a particular line, the substitution list consists of dummies associated with all the macros that the line is in, as well as the dummies associated with the DO blocks that the line is in. The dummies are separated by commas, and there are no spaces in the list. The length of this substitution list should never exceed forty-two characters.

The assembler reads each of the macros into memory before it processes them. There is an upper limit of twelve kilobyes on the total cumulative size of the macro source code.

A convenience of the assembler is its index variable symbol, [XX]. This symbol has an implicit numeric value of 00 to 99. Whenever an [XX] is encountered, the assembler substitutes for it the number of times that the current macro has been called. Each time the macro is called, it is incremented by 1. When the macro is called for the first time, [XX] has a value of 00. This index symbol can be used to tell a macro how many times it has been called, or it may be appended to a generic identifier (of less than 8 characters) to form continually changing labels. This capability is useful when a macro must call itself recursively. Often, when a macro calls itself, the duplication of labels creates confusion and generates error messages. If the index symbol is used and appended to a general name then the labels are unique.

Examples:

[XX]	THE INDEX ITSELF
LOOK[XX]	LOOK01, LOOK02, LOOK03, ETC.

Sample Program Using Macro

Fig. 13 is a listing of a program that uses a macro to examine a register.

Assembler (ASM8) Operating Procedures

ASM8 can have up to two inputs and three outputs. The user must specify the input files. These input files are the source file and an optional macro library file. The outputs are the listing file, the error file, and the cross-reference file. The user can direct the first two of these output files to either the disk, teletypewriter (#TY or #SC), or line printer (#LP). The cross-reference file, however, must be a disk file because ASM8 uses it as an intermediate file for creating the cross-reference table.

The command line consists of the command ASM8 followed by a space, the source filename, and a string of up to four filenames or devices, separated by spaces and followed by a semicolon and string of options. The order of names or devices is macro filename, listing destination, cross-reference listing destination, and error listing destination.

ASM8 <source filename>[,<macro filename>]
[,<listing filename or device>]
[,<xref filename or device>]
[,<error filename or device>]
[;<options>]

The options and defaults specify which of these files or destination devices are necessary. If no options and no filenames are given (except for the source file name) there is no macro file or cross-reference listing, and the listing is sent to the disk with a filename of <source name>.LST:<opposite of source>. The error listing goes by default to the teletypewriter (#TY).

```
ASM NAME.SCR
LISTING - NAME.LST:1
ERRORS - #TY
```

Note: If the cross-reference listing file or the error file is named by the user, the listing file must also be named.

The options specify which of the outputs are to be created, but those that are created must appear in the command line in the order of macro, listing, cross-reference listing, and error listing.

- M - Specifies that a macro file will be used.
- X - Specifies that a cross-reference listing will be created. It will have a default value of <source name>.XRF:<opposite drive from source>
- N - Specifies that a listing will not be created. If this option is not used, the default value will be <source name>.LST:<opposite drive from source>
- H - Specifies that the listing shall contain the hex code only.
- P - Specifies that the assembler should pause after loading to allow the changing of disks.

```
!M
0000 ;                          0001            MACRO
0000 ;                          0002          LOOK [LOOK1]          ..EXAMINE A REGISTER
0000 ;                          0003    ..THIS MACRO ALLOWS EXAMINATION OF A REGISTER
0000 ;                          0004    ..REGISTER RF IS DESTROYED IN THE PROCESS
0000 ;                          0005    ..THE CALLING STATEMENT IS LOOK <REGISTER>
0000 ;                          0006    TYPE EQU 81AEH    ..THE UTILITY TYPING ROUTINE
0000 ;                          0007    TYPE2 EQU 81A4H
0000 ;                          0008    TEMPORARY EQU RF
0000 ;                          0009            [LOOK1].1->TEMPORARY.1
0000 ;                          0010            CALL TYPE ..TYPE THE HI BYTE
0000 ;                          0011          [LOOK1].0->TEMPORARY.1
0000 ;                          0012            CALL TYPE ..TYPE THE LO BYTE
0000 ;                          0013            20H->TEMPORARY.1
0000 ;                          0014            CALL TYPE2 ..TYPE A SPACE
0000 ;                          0015          ENDM
0000 ;                          0016    ..THIS PROGRAM CALLS THE LOOK MACRO TWICE
0000 ;                          0017    ONE EQU 1
0000 ;                          0018    TWO EQU 2
0000 ;                          0019    REGISTER EQU R7
0000 ;                          0020    INIT1 EQU 83F3H
0000 7100C083F3;                0021            DISABLE;IDLE;LBR INIT1 ..INITIALIZE FOR
0005 ;                          0022              ..STANDARD ALL AND RETURN
0005 F803A7F800B7;              0023            (ONE+TWO)->REGISTER.0;0->REGISTER.1    ..PUT 3
                                in R7
000B ;                          0024            LOOK REGISTER              ...EXAMINE R7
000B 97BF ;                     0024    REGISTER.1->TEMPORARY.1
000D D481AF;                    0024    CALL TYPE                          ..TYPE THE HI BYTE
0010 87BF;                      0024    REGISTER.0->TEMPORARY.1
0012 D481AE;                    0024    CALL TYPE                          ..TYPE THE LO BYTE
0015 F820BF;                    0024    20H->TEMPORARY.1
0018 D481A4;                    0024    CALL TYPE2                         ..TYPE A SPACE
001B F801A8F800B8;              0025    ONE->R8.0;0->R8.1                  ..PUT 1 IN R8
0021 ;                          0026    LOOK R8                            ..EXAMINE R8
0021 98BF;                      0026    R8.1->TEMPORARY.1
0023 D481AE;                    0026    CALL TYPE                          ..TYPE THE HI BYTE
0026 88BF;                      0026    R8.0->TEMPORARY.1
0028 D481AE;                    0026    CALL TYPE                          ..TYPE THE LO BYTE
002B F820BF:                    0026    20H->TEMPORARY.1
002E D481A4;                    0026    CALL TYPE2                         ..TYPE A SPACE
0031 00;                        0027    IDLE
0032 ;
0000
```

Fig. 13 - Sample program illustrating use of macros.

B - Specifies that the symbol table will not be initialized and that the symbol table existing in memory will be used.

T - Specifies that the cross-reference listing should be formatted in 80 character lines instead of the default 132 character lines.

Examples:

```
ASM8 MYPROG;P
  SOURCE = MYPROG:0 MACRO = NONE
  LISTING = MYPROG.LST:0
  XREF LISTING = NONE
  ERRORS = #TY
```

The assembler will pause after loading itself to allow for changing of disks.

```
ASM8 MYPROG.S, MAC. M;MH
  SOURCE = MYPROG.S:0 MACRO = MAC.M:0
  LISTING = MYPROG.HEX:1
  XREF LISTING = NONE
  ERRORS = #TY
```

The listing contains only the hex code.

```
ASM8 MYPROG.S, #LP, #LP, #LP;XB
  SOURCE = MYPROG.S:0 MACRO = NONE
  LISTING = #LP
```

XREF LISTING = #LP
ERRORS = #LP

The assembler will not initialize the symbol table.

ASM8 S,M,L,X,E;MXNT
SOURCE = S:0 MACRO = M:0
LISTING = L:1
XREF LISTING = X:1
ERRORS = E:1

All the listings will be in 80-character format.

Fig. 14 summarizes graphically the assembler operating procedures, source, and destination.

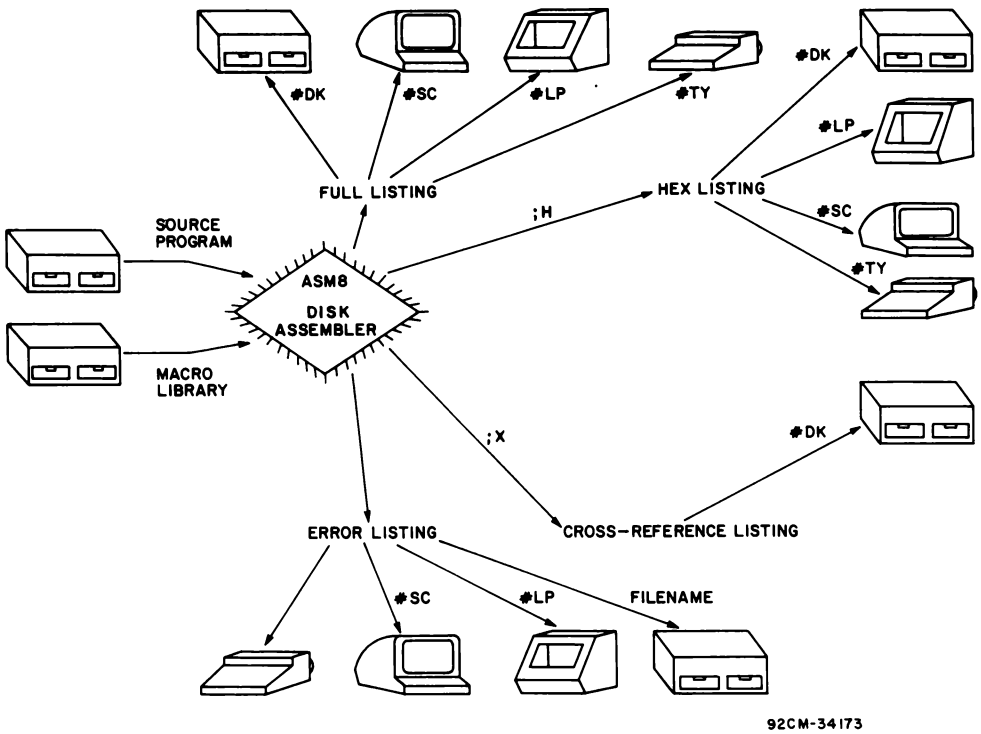


Fig. 14 - ASM8 data flow diagram.

Cross-Reference Listing

The assembler will upon request output the cross-reference table. The first column in the cross-reference listing is the symbol or identifier. Next is its address or value. Third is the line number of the source code where that identifier was defined. The remainder of each line is a list of the lines in the source code where that identifier was referenced.

The cross-reference file can often be useful in locating spelling errors in a program. Fig. 15 is the cross-reference listing from the example program, Figs. 10 and 12. The U in the cross-reference listing indicates that UTILITY was defined as a register and has no address or value.

Error Messages

Non-Fatal Errors

ASM8 will flag **simple errors** and will report the cause of each while it continues to process. Table VIII is a list of these errors and contains suggestions to the user to aid in determining the cause of the errors.

SYMBOL	ADDR	DEF	REFERENCES
ADD_NUMS	000A	0012	0010
ANSWER	0017	0019	0015 0016
FIRST_NUM	0019	0003	0006
SCND_NUM	001F	0004	0012
UTILITY	0008	U	0006 0008 0012 0012 0017

Fig. 15 - Cross-reference listing.

Table VIII - ASM8 Error Messages

1. *** ILLEGAL LABEL - ?????????? ***
   The ?'s are replaced with the label found. Check to see if accidentially a number began in column 1, defining it as a label. Check to see if the label name is a valid-op-code mneomonic.
2. *** DUPLICATE LABEL - ?????????? ***
   The ?'s are replaced with the label found. Check to see if a macro with the same label in it has been called twice. Are two similar labels misspelled?
3. *** ILLEGAL OPERATION - ?????????? ***
   The ?'s are replaced with the op-code found. Is there a misspelled op-code?
4. *** UNDEFINED SYMBOL - ?????????? ***
   The ?'s are replaced with the symbol found. Check for misspelling both at the line flagged and at the definition point.
5. *** ILLEGAL EXPR - ?????????? ***
   The ?'s are replaced with the last ten characters before the error detection point. Check to see if the expression is missing anything such as parentheses.
6. *** BR OUT OF RANGE - ???? ***
   The ?'s are replaced with the paged address. A short branch goes to a point on a different page and must be changed to a long branch.
7. *** ILLEGAL CONST - ???? ***
   The ?'s are replaced with the constant found. Did an identifier begin with a number or is an H or Q on the end of a hexadecimal or octal constant left out?
8. *** OPERAND MISSING ***
   Check the op-code to see how many and what type of operands are required for it. If it is a macro call, check the macro definition for the number of operands required.
9. *** IF STATEMENT ERROR ***
   The expression in the IF statement did not produce a logical true or false. A true result is assumed.
10. *** INVALID REG - ?? ***
    The ?'s are replaced with the number in question. Check the spelling of the identifier and that its value is an addressable register.
11. *** ILLEGAL OPERAND - ?????????? ***
    The ?'s are replaced with the operand in question.

Fatal Errors

Under certain conditions ASM8 will no longer be able to continue processing the source file. For such **"fatal errors,"** the message

ASM ABORTED

will appear on the teletypewriter followed by the conditions causing the abort. These conditions represent system size limitations, and the remedy is a reduction in complexity or size of the source file. They are

SYMBOL TABLE OVFLO
- Too many symbols were defined

WORK AREA OVFLO
- Too complex a DO LOOP was created

MACRO STORE OVFLO
- Too many macros were defined

MACRO DEF ERROR
- There was an incomplete or erroneous macro definition

DO LOOP ERROR
- A DO LOOP was set up incorrectly

In addition to the above, if more than 99 erors are encountered on the first pass, 'ASM ABORTED' will appear with no further explanation. In this case the user need only to attend to the errors already reported and then rerun his assembly.

Warnings

There may also be situations in which the output may appear to be completely correct but probably is not. In these cases ASM8 will issue warning messages to the teletypewriter. These warning messages are

X-REF TABLE OVFLO
- The cross-reference listing is incomplete (more than 6144 references)

DUPLICATE MACRO NAME
- Macro expansions may be incorrect

LOC CTR ERROR
- The final values of the location counter after each pass were different

7. MicroDOS User Functions

The set of MicroDOS User Functions that can be called directly from an application program is a significant feature of the MicroDOS Operating System. In this chapter, the uses of the specific functions are described. It is important, however, to have an understanding of two basic concepts, the I/O Control Block and the Buffers, before the user functions can be utilized.

I/O Control Block and Buffers

The IOCB (I/O Control Block) is a depository of information for the I/O channel through which the user is communicating. An IOCB is a software analog of the hardware interface boards found in any computer system. One IOCB must be set up for each channel of communication. Thus, a standard data terminal would have two IOCB's associated with it; one for characters received from the keyboard, and another for information sent to the terminal for display. Reading a disk file requires a single IOCB; reading from and writing to a disk file requires a total of two.

For some user function routines such as TYPE (which outputs characters to the terminal) the IOCB is already set up and the user need not be concerned with it. The appropriate IOCB for TYPE was set up previously because the MicroDOS operating system is already in communication with the terminal. A separate IOCB, however, will have to be set up for any disk reading or writing that is wanted.

The second important concept is the buffer. A buffer is simply a reserved block of RAM through which data is passed on its way to and from the I/O devices. The CREAD routine, for example, is structured to imput data to the buffer as it is received from the keyboard. Later the input characters can be examined and acted upon by the user's program. Similarly, the TYPE routine picks up data bytes from a specified buffer area and outputs them to the terminal. Disk I/O is handled a sector at a time (512 bytes) and is similarly passed through a buffer. A part of the information in the IOCB is the two addresses specifying the start and the termination of the sector buffer. Buffer areas must be reserved for all I/O operations through MicroDOS. For disk IOCB's, the reserved area must be 512 bytes in length.

When an error occurs, the state of the IOCB is indeterminate.

IOCB Initialization

Fig. 16 shows the structure of how an IOCB is initialized. A description of each area follows.

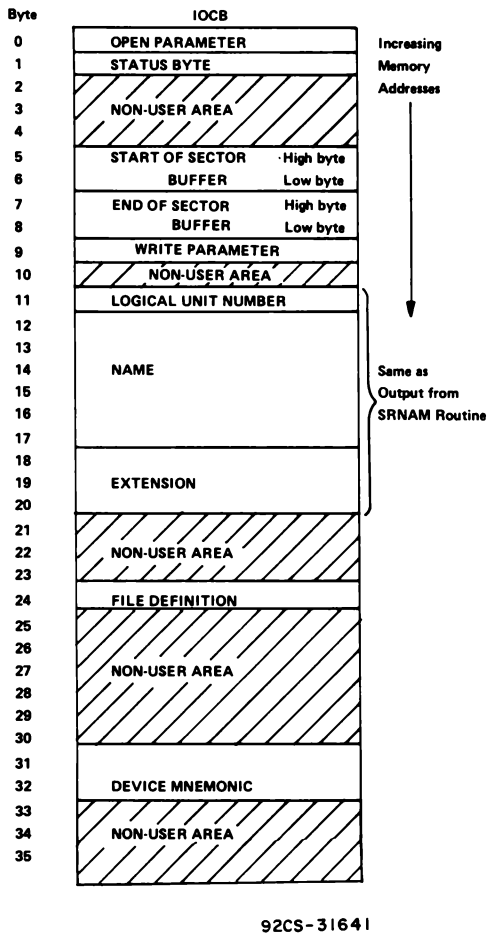


Fig. 16 - Diagram of Input/output Control Block (IOCB) Structure.

Byte 0 - Open Parameter. Any file or I/O device can be opened for reading or writing. The value of byte 0 specifices which operation is to be performed. For READ the appropriate value is B1H. For WRITE the appropriate value is 7AH. If the value 7BH is used, a new file will be opened if one does not already exist. Otherwise, it will open the existing file for writing.

Byte 1 - Status Byte. When a user function is called, it places a value in byte 1 to indicate whether or not the

operation requested was successful. A zero indicates success. Non-zero numbers are coded erro-message representations. Appendix D provides a listing of the error-message numbers and their meanings. In addition, the value C9H will be placed in byte 1 when an end-of-file marker has been read. The appropriate message can be automatically written to the terminal by calling the user function CDERR, which will be discussed later.

Bytes 2 to 4 - Non-User Area. This area, as well as bytes 10,21 to 23, 25 to 30, and 33 to 35, is not available to the user.

Bytes 5,6 - Start of Sector Buffer. In bytes 5 and 6, the user enters the starting (lowest value) address of the associated buffer. The high byte is entered in 5 and the low byte in 6.

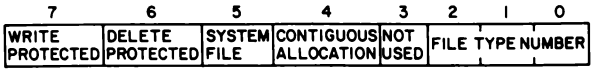
Bytes 7,8 - End of Sector Buffer. In bytes 7 and 8, the user enters the last (highest) address of the buffer. The high byte is entered in 7 and the low byte in 8. For disk IOCB's the buffer length must be 512 bytes. For other input devices the buffer length should be the maximum number of data bytes to be received plus one. For other output devices the buffer length is equal to the length of the maximum number of bytes to be transmitted.

Byte 9 - Write Parameter. When a disk file is opened for writing, byte 9 defines the number of clusters to be allocated for the file (a custer = 1 sector). The standard allocation of 27 clusters is denoted by zeros in this byte. Any non-zero values denote the number of clusters: 1,2,3,...etc. Because additional space will be automatically allocated as needed, it does not matter if the file size is not known. An attempt to over-allocate to accommodate the largest possible file may result in a "DISK FULL" indication when, in actuality, the file might fit.

Byte 11 - Unit Number. Byte 11 is set to '0' for the left disk drive or to '1' for the right one. It normally should be set to zero as the default value. If a drive is specified as part of the file name (as in NAME.EXT:DRIVE#), the user function SRNAM will put the drive # in this byte.

Bytes 12 to 20 - Name and Extension. The six-byte name and the three-byte extension (stored in ASCII) is the name associated with a disk file. Again, the SRNAM routine can be used to fill in these bytes. For non-disk IOCB's bytes 11 to 20 have no meaning. Note: This area must be initialized with the ASCII 'space' character (20H) each time before SRNAM is called.

Byte 24 - File Definition. Byte 24 defines the disk file type (binary or ASCII) and attributes. MicroDOS system files are all of the binary type; in general, user-generated files are ASCII. Attributes occupy various bit positions as given in Fig. 17. The attribute is enabled when the bit is set to '1'.



BITS 2 1 0	FILE TYPE
0 0 0	INTERLEAVED BINARY
0 0 1	BINARY
0 1 0	ASCII/ASCII-HEX
0 1 1	OPERATING SYSTEM
1 0 0	INTERLEAVED ASCII/ ASCII-HEX

92CS-34178

Fig. 17 - Attitude bit positions.

Bytes 31, 32 - Device Mnemonic. Five different device mnemonics are presently supported by MicroDOS. The user should enter one of the pairs of characters given below in ASCII code into these two bytes. The SRNAM routine can be used to fill in these bytes. The default value of DK, however, should be entered by a user program.

DK identifies the disk for both input or output IOCB's
LP identifies the line printer for output
TY identifies a teletypewriter for output
KB identifies the console keyboard for input
SC identifies the console video screen for output

IOCB Changes After a File Is Opened

A file must be opened before any disk read or write operation can take place. The routine OPEN is used for this purpose. OPEN is described in detail in the next section. OPEN, however, changes many values of the initialized IOCB in order to set up various pointers. Specifically, the following alterations are made:

Bytes 5 to 8 - Sector Buffer. OPEN will use the buffer area indicated by these bytes and over-write any data already there.

Byte 0 - Open Parameter. Bit 4 of this value is reset to 0 when a file is openend. Thus, for read operations the value becomes A1H and for write it becomes 6AH. Bit 4 is set to 1 when a file is closed.

Byte 9 - Write Parameter. This value is replaced by a pointer to the present position in the Sector Buffer.

Bytes 11 to 20 - Unit Number, Name, and Extension. These values are replaced by pointers to the disk file.

Bytes 31, 32 - Device Mnemonic. This area becomes a pointer to the appropriate code in the MicroDOS operating system for the device I/O operation.

IOCB Example

As an example of a complete initialized IOCB, Fib. 18 shows one set up for reading the ASCII disk file called TOM,CHK on drive zero. Any number of files

can be open simultaneously, limited only by available RAM.

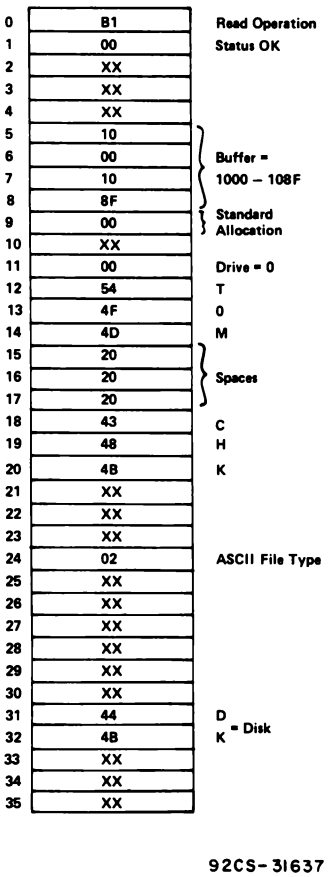


Fig. 18 - Typical IOCB for reading a disk file named TOM.CHK.

Introduction to User Function

In this section the MicroDOS functions that the user can call directly from an application program are described. These functions, among other things, allow the user to read or write to and from disk files. Some of these functions are conveniences to facilitate setting up the IOCB. Others are called to do the actual I/O operations in a way analogous to the UT71 READ and TYPE routines (See next chapter, Utility Program UT71). The MicroDOS console read and type routines themselves use UT71 READ and TYPE to do byte I/O transfers. MicroDOS console routines, however, are designed to operate on buffers of data rather than a byte at a time.

The general form for calling a user function is:

CALL UCALL,<FN>,[<PARMn>]

where: CALL EQU OD4H (The assembler will do the translation)

UCALL EQU OB453H

<FN> is the value assigned to the function

<PARMn> are parameters passed to the called routine

The Standard Call and Return Technique (SCRT) must be adhered to when these conventions are used.

The conventions are as follows:

R2=stack pointer
R3=program counter
R4=address of CALL routine ¢
R5=address of RETURN routine ¢
R6=pointer to return point

Most CPU registers are preserved during a call to a user function (saved and then restored). Up to 52 bytes of the stack are required for a call.

Console I/O Routines

1. Function: **CREAD**

2. Value:12H

3. Description:

CREAD is used to read a line from the console device into a buffer.

4. Format:

CALL UCALL,CREAD,BUFFER,BYTECT

where: BUFFER is the starting address of the RAM buffer into which the data is to be put. Its length must be BYTECT + 1, where BYTECT is the number of characters to be input. BUFFER will contain the entered characters plus the terminating carriage return from low-to-high address. With the exceptionof (CR), the following characters (RUBOUT, CANCEL, and LINE FEED) are handled as special control functions and are not put into the buffer.

RUBOUT (7FH): When a RUBOUT is pressed, a left bracket "[" is printed followed by the deleted character. When a NON-RUBOUT is pressed, a right bracket "]" is printed followed by the pressed character. The RUBOUT deletes the last character in the buffer thus providing a built-in line-editing function.

CANCEL (CTRL-C): Deletes all characters in the buffer and awaits the next character.

LINE FEED: Displays the contents of the buffer on the next line and awaits the next character.

CARRIAGE RETURN: Terminates input. (This character is put into the buffer and causes a carriage return and line feed.)

Before typing:

Increasing addresses ⟶

Buffer [ ]

After typing "ABCD(CR)":

Buffer [ABCD(CR)]

5. Example:
Input a line of up to 20 characters in a buffer starting at location 1000H.

```
UCALL EQU 0B453H
CREAD EQU 12H
BUFFER EQU 1000H

    .
    .
    .
CALL UCALL,CREAD,BUFFER,21
```

1. Function: **TYPE**
2. Value: 14H
3. Description:
TYPE outputs the defined text to the terminal.
4. Format:
CALL UCALL,TYPE,BUFFER
where: BUFFER contains the data to be typed.
Typing will be terminated by a null (00H) character in the buffer. Data will be output from low to high addresses.
5. Example:

```
UCALL EQU OB453H
TYPE EQU 14H

    .
    .
CALL UCALL,TYPE,MSG1
    .
    .
MSG1 DC ODOAH,'MICRODOS TEST PRO-
GRAM', OOH
```

Disk I/O Routines

1. Function: **GETCHR**
2. Value: O8H
3. Description:
This routine reads a character from an opened file and returns the character in RF.1.
4. Format:
CALL UCALL,GETCHR,IOCB
where: IOCB has previously been opened. See the OPEN function. The status byte of the IOCB will be updated by this routine and should be checked for an error. If the status byte is non-zero, CDERR should be called to print the error message.

5. Example:
Read a character from an opened file and check for an end-of-file marker.

```
UCALL EQU OB453H
CDERR EQU 28H
GETCHR EQU 08H
STATUS EQU 9 ..R9 contains IOCB + 1
    .
    .
CALL UCALL,GETCHR,IOCB
                ..Read byte.
LDN STATUS      ..Check status byte
LBNZ ERROR      ..Branch to error routine, else
GHI RF          ..Get the character
XOR 13H         ..it is a 'DC3'?
LBZ END         ..If so, go to END
                ..If not, this is the next
                ..instruction
    .
    .
IOCB DC OB1H    ..This is the first byte of the
                ..IOCB for the file being
                ..read
    .
    .
ERROR CALL UCALL,CDERR,IOCB
                ..Display error message
    .
    .
```

1. Function: **PUTCHR**
2. Value: OEH
3. Description:
This routine outputs a character to an opened file. The character must be placed in RF.1 before the routine is called.
4. Format
CALL UCALL,PUTCHR,IOCB
where: IOCB has been previously opened. The status byte of the IOCB will be updated. After calling this routine, the user should call CDERR to print any error messages.
The last character output for most ASCII files should be DC3 (13H), the end-of-file marker. Then, the PUTSEC user function must be called before the file is closed. This call assures that the last 512 bytes will be written on the diskette.
5. Example:
Close the disk file being writen to.

```
UCALL EQU OB453H
PUTCHR EQU OEH
CDERR EQU 28H
CLOSE EQU 02H
STATUS EQU 9 ..R9 contains IOCB + 1
    .
    .
LDI 13H; PHI RF..Output end-of-file marker
CALL UCALL,PUTCHR,IOCB
LDN STATUS          ..Check status byte
LBNZ ERROR
```

¢ *As described in the* **User Manual for the CDP1802 Microprocessor,** MPM-201.

```
CALL UCALL,PUTSEC,IOCB
                    ..Write out last sector
LDN STATUS          ..Check status byte
LBNZ ERROR
CALL UCALL,CLOSE,IOCB
                    ..Close file
LDN STATUS          ..Check status byte
LBNZ ERROR
        .
        .
ERROR CALL UCALL,CDERR,IOCB
                    ..Display error message
        .
        .
IOCB DC 7AH         ..This is the first byte of the
                    ..IOCB for the file.
```

1. Function: **GETSEC**

2. Value: 06H

3. Description:

The GETSEC routine causes one sector (512 bytes) to be read from the opened file into the sector buffer described by the IOCB.

4. Format:

CALL UCALL,GETSEC,IOCB

where: IOCB is associated with the opened file.

After each call to this routine, MicroDOS sets up the IOCB so that the user can read from the next consecutive sector. The status byte of the IOCB will be updated. After calling this routine, the user should call CDERR to print any error messages. This utility is not required under normal conditions because consecutive calls to GETCHR will automatically advance to the next sector every 512 bytes. It is included as a convenience for those wishing to write their own special programs and keep their own byte count. If the user wants to randomly access a logical sector in a file, he can change bytes 19 and 20 in the IOCB so that they equal the desired logical section before the call to the routine is made.

5. Example:

Search an opened file for the first sector containing a NULL as the first character.

```
    UCALL EQU OB453H
    GETSEC EQU 06H
    GETCHR EQU 08H
    STATUS EQU 9   ..R9 contains IOCB + 1
        .
        .
LOOP CALL UCALL,GETSEC,IOCB
                    ..Point to next sector
LDN STATUS;LBNZ ERROR
                    ..Check status
CALL UCALL,GETCHR,IOCB
                    ..Get first character
LDN STATUS;LBNZ ERROR
                    ..Check status
GHI RF;BNZ LOOP
                    ..Loop back if not = 00
        .
        .
IOCB DC OB1H        ..This is the first byte of the
                    ..IOCB for the file being read
        .
        .
ERROR CALL UCALL,CDERR,IOCB
                    ..Display error message
```

1. Function: **PUTSEC**

2. Value: 10H

3. Description:

The PUTSEC routine causes one sector (512 bytes) to be written to the opened file from the sector buffer described by the IOCB.

4. Format:

CALL UCALL,PUTSEC,IOCB

where: IOCB is associated with the opened file.

After each call to this routine, MicroDOS sets up the IOCB so that the user can write the next consecutive sector. The status byte of the IOCB will be updated. After calling this routine, the user should call CDERR to print any error messages.

If disk transfers are being done on a character basis, this routine should be called after the last byte (the end-of-file marker DC3) is output to a file to make sure that the last 512 bytes actually get written on the diskette. See the example under PUTCHR. If the user wants to randomly access a logical sector in a file, he can change bytes 19 and 20 in the IOCB so that they equal the desired logical section before the call to the routine is made.

1. Function: **CLOSE**

2. Value: 02H

3. Description:

The CLOSE routine performs all the necessary functions after a file has been used.

4. Format:

CALL UCALL,CLOSE,IOCB

where: IOCB relates to the file that is to be closed. The status byte of the IOCB must be checked after each CLOSE operation by calling CDERR. The CLOSE function does not write out any partially filled sectors nor does it add DC3 as the last character in the file. Its main function is to deallocate disk space no longer required. See the example under PUTCHR.

1. Function: **OPEN**

2. Value: OOH

3. Description:

The OPEN function prepares a file for subsequent use.

4. Format:

CALL UCALL,OPEN,IOCB

The IOCB must be initialized before a file is opened. Attempting to read or write to an unopened file will cause errors. A call to OPEN will change almost all areas of the IOCB from their initialized values. The status byte of the IOCB will also be updated by this routine. After calling this routine, the user should call CDERR to display any error messages.

Note: OPEN uses the buffer area pointed to by the IOCB. OPEN, therefore, should be called before valid data is accumulated in an output buffer.

5. Example:

Open a file for which the IOCB has been set up and read the first character.

```
    UCALL EQU OB453H
    OPEN EQU OOH
    GETCHR EQU O8H
STATUS EQU 9          ..R9 contains IOCB + 1
        .
        .
    CALL UCALL,OPEN,IOCB
                      ..Open file
    LDN STATUS;LBNZ ERROR
                      ..Check status
    CALL UCALL,GETCHR,IOCB
                      ..Get first character
        .
        .
    IOCB DC OB1H      ..This is the first byte of the
                      IOCB
        .
        .
    ERROR CALL UCALL,CDERR,IOCB
                      ..Display error message
```

1. Function: **REWIND**

2. Value: 04H

3. Description:

The REWIND function positions the IOCB pointer to the beginning for the file.

4. Format:

CALL UCALL,REWIND,IOCB

where: IOCB relates to the file that is to be "rewound". After this routine is called, the next character read will be the first character of the file.

1. Function: **CDERR**

2. Value: 28H

3. Description:

The CDERR routine displays a pertinent error message from the library of error messages.

4. Format:

CALL UCALL,CDERR,IOCB

where: IOCB is the Input Output Control Block containing the error number in its status byte.

After a user function requiring an IOCB as a parameter is called, that function returns in the status byte a zero for no error or a non-zero value which identifies an error. See Appendix D for a complete listing of MicroDOS error messages with their identifying numbers and meanings. The CDERR function displays the correct error message for the error condition.

5. Example:

Read a byte from an opened file and check for an error condition. Register R9 will be used as a pointer to the status byte.

```
    UCALL EQU OB453H
    GETCHR EQU O8H
    CDERR EQU 28H
STATUS EQU 9          ..R9 contains IOCB + 1
        .
        .
    CALL UCALL,GETCHR,IOCB
                      ..Get character
    LDI A.O(IOCB + 1);PLO STATUS
                      ..Point R9 to status byte
    LDI A.1(IOCB + 1);PHI STATUS
    LDN STATUS; LBNZ ERROR
                      ..Get status and check
        .
        .
    ERROR CALL UCALL,CDERR,IOCB
                      ..Display error message
        .
        .
IOCB DC OB1H          ..This is the first byte of
                      ..the IOCB
        .
        .
```

IOCB Setup Aid Routine

1. Function **SRNAM**

2. Value: 24H

3. Description:

The SRNAM (Search-for-File-Name) routine searches a specified input buffer for a file name, and then reformats and moves the information to the appropriate area of an IOCB. It is designed to help in setting up an IOCB by taking file name information from a line buffer (put there by CREAD) and relocating it into an IOCB.

4. Format:

CALL UCALL,SRNAM,PACKET

where: PACKET is a special 4-byte pointer in which the first two bytes point to the input buffer and the second two bytes point to the unit number byte in an IOCB. Fig. 19 depicts the operation of SRNAM.

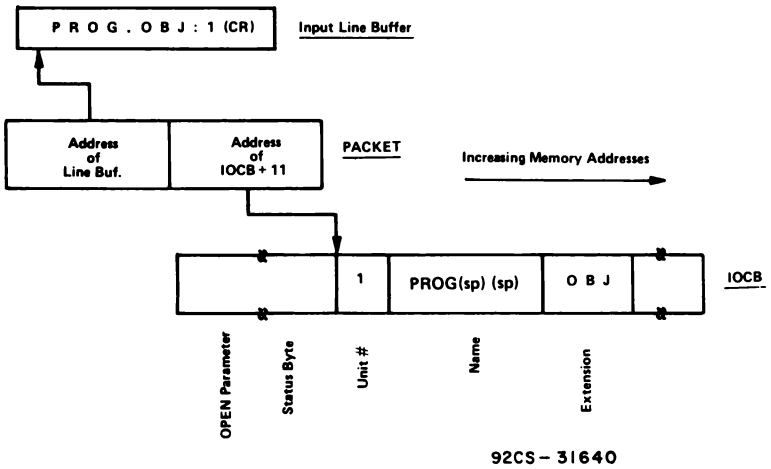


Fig. 19 - Pictorial representation of SRNAM operation.

SRNAM maintains a status word (located at B452H) to indicate the results of its operation. A valid file name found is indicated by 00H in this byte. The setting of the various bits have the following meanings.

Bit 0=1; an asterisk * (wild-card) was found in the file name.

Bit 1=1; an * was found in the extension.

Bit 2=1; a device name (LP,TY,SC,KB, or DK) was found instead of a file name. The device mnemonic will be placed in the proper area of the IOCB.

Bit 7=1; no file name was found.

SRNAM may be called repeatedly to pick up a series of file names from an input buffer and place them in various IOCB's. The IOCB-pointer part of PACKET must be changed each time to perform this operation, but the input-buffer-pointer section of PACKET is automatically positioned past each file name as it is encountered. SRNAM makes no changes to the input buffer. SRNAM returns to the caller after each file name is encountered. However, it will not search past a semicolon or carriage return.

SRNAM will not place delimiters in the file name area, and spaces encountered before the file name will not be used. Any characters found after the maximum allowed for a field will be discarded. For example, if eight characters are used for the file name, only the first six will be placed in the output buffer. The output data is changed only if that area was encountered in the search. Before calling SRNAM, therefore, the IOCB should be initialized to the desired default values by setting the Unit Number = 00, filling the NAME and EXT areas with 20H (ASCII space), and setting the Device Mnemonic area to DK (ASCII).

5. Example:

By means of CREAD, two file names have been entered into a line buffer (BUF1). Using SRNAM, put the names into IOCB1 and IOCB2.

BUF1 contains

AB12.M, XYX.N:1 (CR)

PACKET is set up as follows:

ADDRESS OF BUF1
ADDRESS OF IOCB1 + 11

IOCB1 + 11 and IOCB2 + 11 were initialized as follows:

O	(sp)(sp)(sp)(sp)(sp)(sp)	(sp)(sp)(sp)

After the first call to SRNAM, PACKET will look like:

ADDRESS OF BUF1 + 7
ADDRESS OF IOCB1 + 11

and OICB1 looks like:

O	AB12(sp)(sp)	M(sp)(sp)

Next, PACKET is reinitialized to point to IOCB2 and looks like:

ADDRESS BUF1 + 7
ADDRESS OF IOCB2 +11

A second call to SRNAM makes PACKET look like:

ADDRESS OF BUF1 + 17
ADDRESS OF IOCB2 + 11

and IOCB2 looks like:

1	XYZ(sp)(sp)(sp)	N(sp)(sp)

Note how SRNAM updates the input address so that the user can keep calling SRNAM to find a series of file names.

Return to MicroDOS Operating System Routine

1. **Function:** CDENT
2. **Value: 1EH**
3. **Description:**

The CDENT routine returns program control to the MicroDOS operating system. The ≥ prompt will be output to the terminal.

4. **Format:**

CALL UCALL,CDENT

This function, rather than an LBR 9000H, should be used to return control to the operating system. Note that this function does not close files, update the director, or save the CPU status.

Operating Sequence Summary

The following is a summary of the steps necessary to do disk I/O with MicroDOS user functions.

1. Reserve buffer areas:
   512 Bytes for a disk channel
   80 bytes or less for keyboard input.
   4 bytes for SRNAM packet.
2. Set up as many IOCB's as required. Set the OPEN parameter for read or write. Fill in the sector buffer pointers.
   Set Unit #=0 (for default) or as required.
   Fill file name and extension areas with 20H or name, if fixed.
   Set file definition.
   Fill in device mnemonic = DK (for default) or as required.
3. Set up PACKET pointing to input buffer and IOCB.
4. Call CREAD to input file name, if a variable.
5. Call SRNAM to move file name to IOCB. Check status byte at B452H.
6. Call OPEN. Check status byte of IOCB. If non-zero, call CDERR to output the error message and reinitialize the IOCB.
7. Call GETCHR or PUTCHR to do disk read or write. Check status byte of IOCB. If non-zero, call CDERR to output the error message.
8. When writing is finished, output 13H (end-of-file marker) and call PUTSEC. Check status byte of IOCB. If non-zero, call CDERR to output the error message.
9. Call CLOSE. Check status byte of IOCB. If non-zero, call CDERR to output the error message.
10. To return to the MicroDOS operating system, call CDENT.

A sample program illustrating the use of user functions is given in Appendix E.

8. Monitor Programs UT71

The Monitor Program UT71 enables the user to examine or alter memory, begin program execution at a given location, do I/O from the keyboard, or transfer data between disk and memory. In addition, it can set up half- or full-duplex operation, load the operating system, or perform a test on itself. These functions are accomplished through a series of monitor commands that are initiated by typing D, F, I, M, S, P, T, L, B, ?, !, R, or W. The functions include memory display (D), memory fill (F), memory insert (I), memory move (M), memory sutstitute (S), run program (P), self test (T), load operating system (L or B), do I/O from keyboard (? or !), and disk read (R) or write (W). Also included are the standard read and type routines that provide communication with the user's terminal. Finally, the monitor contains routines that communicate with the RCA MSIM 50 3½-inch micro floppy disk drives through the CDP18S651 disk controller.

After the system is powered up, the monitor issues an asterisk prompt "*" indicating that it is ready to accept monitor commands. Pressing RESET/RUN U will also result in the same prompt.

Register Save

When the system is started from RESET/RUN U, the contents of the CPU registers are saved in RAM at 8C00H. The contents of R0 and R1 however, are destroyed by the process. The contents of the saved registers can be examined by displaying memory at 8C00H for 20 bytes. This register-save feature can be used to debug machine-language programs. First, insert an IDLE instruction (00) in the program code at the appropriate place. Next, execute the program and wait until the IDLE is reached. Then press RESET/RUN U and examine memory at 8C00H to determine the contents of the registers at the registers at the time the IDLE was encountered.

Self Test

The user can start the self-test function from the monitor by typing a T.

The test will perform an 8-bit checksum of the UT71 PROM. The results should be zero. If not, the system will print:

PROM BAD

Next, it will perform a read/write test on all RAM. It starts at 8800H and wraps around, ending at 7FFFH. If a bad location is found, the test ends and prints:

RAM BAD, P(PAGENO)

If all the tests pass, the following will be printed:

MEMORY OK

When the self test is finished, control is returned to the monitor.

UT71 Commands

Following is a description of the UT71 commands. Note that all address, data, and byte counts are entered as hexadecimal numbers. In the examples given, the characters generated by the system are underlined. The monitor prompt is an asterisk *.

T Command

Name: **Test**
Purpose: Memory self test
Format: T
Action: Tests all ROM and RAM
Example T
MEMORY OK
*
—

D Command

Name: **Memory Display**
Purpose: To allow a specified area of memory to be displayed on the user therminal.
Format: D(START ADDRESS)(OPTION)(CR)
Action: The contents of memory, beginning at the specified (START ADDRESS) will be transmitted to the user terminal. (OPTION) allows the transmission of either a specific number of bytes preceded by a space or an inclusive address range preceded by a hyphen. If the option is not specified, a default value of 1 byte results.
Examples: D42F8 8(CR)
D42F8-42FF(CR)
Both of these examples produce the same output.

I Command

Name: Memory Insert
Purpose: To alter the contents of memory beginning at the specified address.
Format: I(START ADDR)(SPACE) (DATA)[(CONT)](CR)
Action: A memory location is accessed at the specified (START ADDR). The (DATA) required is one byte specified by two hex digits. The (CONT) option allows data to be continued onto the next line on the terminal with or without changing the current memory address. A (COMMA) will not change the address and after the user inserts (CR)(LF), additional data may be entered. If a (SEMICOLON) is entered and after a user-inserted (CR)(LF), a new address is anticipated. The semicolon allows non-contiguous memory to be loaded with a single insert command. The command may be terminated at any point by the entry of a (CR) not preceded by a (COMMA) or (SEMICOLON).
Examples: I42F8 7100F840B0F88CB1 (CR)

142F8 7100F840,(CR)(LF)
B0F8,(CR)(LF)
8CB1(CR)

142F8 7100F840B0;(CR)(LF)
43B6 94FB903A0F(CR)

The first and second examples give identical results. The second provides improved readability at the data terminal output. The third example enters data into two memory areas, starting at 42F8 and 43B6.

M Command

Name: **Memory Move**
Purpose: To move a block of data from one area of memory to another area.
Format: M(SOURCE ADDR)(OPTION)(SPACE) (DEST ADDR)(CR)
Action: Data is copied from memory source location beginning at the (SOURCE ADDR) into locations specified by the (DEST ADDR). (OPTION) allows the transfer of either a specific number of bytes preceded by a space or an inclusive address range preceded by a hyphen. There is no restriction on the direction of the move and the areas may overlap.
Examples: M42F8 8 43F8(CR)
M42F8-42FF 43F8(CR)
M43B0-43BF 42B0(CR)
M43B0-43BF 43B2(CR)

F Command

Name: **Memory Fill**
Purpose: To load a defined area of memory with a specific constant.
Format: F(START ADDR)(OPTION)(SPACE) (DATA)(CR)
Action: The specified (DATA) is loaded into memory beginning at the (START ADDR). (OPTION) allows the loading of either a specified number of bytes preceded by a space or an inclusive address range preceded by a hyphen.
Examples: F42F8 8 00(CR)
F42F8-42FF 00(CR)
These examples fill with zeros the eight bytes beginning at location 42F8.

S Command

Name: **Memory Substitute**
Purpose: To display and, if desired, alter the contents of sequential memory locations beginning at the specified address.
Format: S(START) ADDR)(OPTION)(CR)
Action: A memory location is accessed at the specified (START ADDR). Its contents will not be displayed, however, until (OPTIONS) is entered. (OPTIONS) allows two methods of display. If (SPACE) is entered, the current data will be displayed on the same line followed by a hyphen. New data may be entered at this point. Only the last byte entered will be written. If no data is entered, the current data will remain unchanged. If a (LF) is entered, a (CR)(LF) will result and the current memory address will be echoed to the terminal prior to the printing of current data. New data may be entered as described above. The command can be terminated by a (CR) or continued by the entry of any of the OPTIONS).
Examples: S42F8 63-71 00- 0F-C0(CR)
The current data of 63 is changed to 71. The 00 data is retained, and the OF is changed to C0
S42F8 71- 00- C0- 11-82(LF)
42FC 52-AE(LF)
42FD 00-F8 11-40 23-A3(CR)
In this example, the 71, 00, and C0 are retained and the 11 is changed to 82. Each (LF) causes the next address to be followed by its data.

P Command

Name: **Program Run**
Purpose: To allow a user program to be run beginning at the specified address.
Format: P[(START ADDR)](CR)
Action: The user program will begin execution at the specified (START ADDR) with P = 0 and X = 0. If the (START ADDR) is not specified, the default value is 0000.

L Command

Name: **Load**
Purpose: Loads the operating system from drive 0.
Format: L
Action: MicroDOS gets loaded into memory from drive 0.
Example: L
MICRODOS 0.0
≥

B Command

Name: **Boot**
Purpose: Loads the operating system from any drive (0-3).
Format: L(drive No.)
Action: MicroDOS gets loaded into memory from specified drive.
Example: B 1
MICRODOS 0.0
≤

R Command

Name: **Read Sector**
Purpose: Transfers one sector of data from disk to memory
Format: R A=(address)(space)D= (drive)(space)T =(track)(space)S =(sector)(CR)
Action: One sector (512 bytes) of data is transferred from the specified disk, track, and sector to memory starting at the specified address. Drive number must be from 0 to 3, track from 0 to 45 hex, and sector from 1 to 9. All defaults are to 0.

W Command

Name: **Write Sector**
Purpose: Transfers one sector of data from memory to disk.
Format: W A=(address)(space)D =(drive)(space)T =(tract)(space)S=(sector)(CR).
Action: This command performs the complement of the R command.

? Command

Name: **Read I/O Port.**
Purpose: Transfer on byte of data from input port to screen.
Format: ? G =(group no.)(space)P =(port no.)(CR).
Action: One byte of data from group address and port number specified is printed on the screen.

! Command

Name: **Write to I/O Port:**
Purpose: Transfer one byte of data from keyboard to output port.
Format: ! G =(group no.)(space)P =(port no.) (space)B =(data)(CR).
Action: One byte of data is output to the group address and port specified.

9. Terminal Interfacing

UART Action

Terminal interfacing is handled by UT71 by means of a UART. TYPE routines in UT71 test to see that the holding register of the UART transmitter is empty and if so, pass the byte to be typed to the UART and then return program control to the caller. READ routines test the Data Available signal from the UART, and when that signal is true, a byte is picked up and returned to the caller. The UART's control register is initialized by UT71 for the serial format consiting of one start bit, eight data bits, and two stop bits, as illustrated in Fig. 20. User programs may change the control word, if desired.

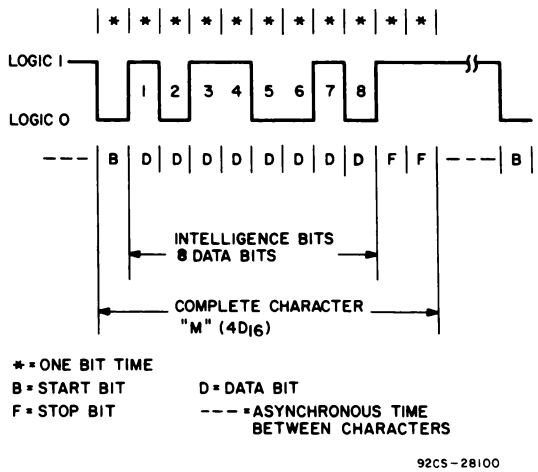


Fig. 20 - Data terminal bit serial output for the character "M".

Refer to Appendix F for the I/O Group 1 assignments for the UART.

ASCII Coding

The system is designed to interface to a data terminal via a serial ASCII code using an EIA RS232C standard electrical interface. When a key is struck on a terminal, the information denoting that character is converted to its ASCII code and appears on the output terminals as a serial data-bit stream. The serial data from the central processor for the letter 'M' is shown in Fig. 20. The character is framed by a start bit B and two stop bits FF. By convention two stop bits are used for data transmission at 10 characters per second although 1, 1-1/2, or 2 are also acceptable outputs from various data terminals.

UT71 Routines READ, TYPE, and OSTRNG

The UT71 READ and TYPE routines provide the basic software mechanism for communication between the system and the data terminal. Several different routines are available to facilitate different types of I/O data transfers.

Register Use

All READ and TYPE routines use R3 as their program counter and return to the caller with SEP R5. They can be called directly from a program that can use R5 as its program counter, or they may be called through the Standard Call and Return Technique (SCRT) described in the **User Manual for the CDP1802 Microprocessor,** MPM-201 in the Section "Programming Techniques" under the heading "Subroutine Techniques." This programming technique is the most general and is recommended.

The upper half of register RE (RE.1) holds a control constant. The least significant bit specifies whether or not characters read in should be "echoed" (full-duplex) or not echoed (half-duplex). A zero in the LSB specifies echo, a 1 specified no echo. UT71 initializes RE.1 to zero for full-duplex operation. If the first character read by UT71 after its initialization is a Line Feed character, the value in RE.1 will be changed to a '1'. Otherwise, operations will proceed with RE.1 = 0.

The most significant bit of RE.1 specifies whether the Command File Interpreter is in control. If set, UT71 will branch to the Interpreter to spot the character. It is very important to always restore RE.1 before doing any read routine.

Two bytes of RAM are needed by the READ and TYPE routines. These routines assume that R2 points to free RAM and M(R(2)) is altered by them. In general, the user can set R2 to any free RAM location. UT71 uses a byte in its dedicated RAM for this purpose.

RF.1 is used in certain cases to pass the byte being read or typed between the calling routine and these subroutines. When READ is exited, it leaves the input byte in RF.1. When TYPE is entered at location 81A4, the byte to be typed is taken from RF.1.

All routines alter RE.0 and RF.0. They also alter D, DF, and X. The READ routine leaves the input byte in D as well as in RF.1 if CALL and RETURN subroutines of UT71 are used. But the byte in D will be destroyed if the Standard Call and Return Technique, described in MPM-201, is used.

READ

When READ exits, R3 is ready for entry at READAH (see Table IX). When TYPE exits, R3 is ready for entry at TYPE5 (see same table).

The READ routine has two entry points - READ and READAH. The former acts as described above and has no other side effects. The latter operates just as READ does, but with the following side effect. If the character read in is a hex character (0-9, A-F) then the 16-bit contents of RD are shifted four bits to the left, and the 4-bit hex equivalent of the input character is entered at the right. DF is then set to 1 on exiting. If the input character is not a hex character, RD is not affected, but DF is set to 0 on exiting.

Table IX - UT71 Utility Routines

Entry Name	Absolute Address	Function
READ	813E	Input ASCII - -> RF.1 (if non-standard linkage)
READAH	813B	Same as READ. If hex character, DIGIT - -> RD (see text)
TYPE5	81A0	Output ASCII Character at M(R5). Then increment R5
TYPE6	81A2	Output ASCII character at M(R6). Then increment R6
TYPE	81A4	Output ASCII character in RF.1
TYPE2	81AE	Output hex digit pair in RF.1
OSTRNG	83F0	Output ASCII string at M(R6). Data byte 00 ends typeout
CKHEX	83FC	RF.1(ASCII) -> RE.0 (hex) and RD.0 (hex); DF =1 if hex, DF =0 if not hex.
INIT1	83F3	Initialize R2, R3, R4, R5, X, P
INIT2	83F6	Initialize R2, R4, R5, X, P
GOUT71	83F9	Return to UT71
LINEPR	850E	Output RF.1 to line printer port
CALLR	8364	SCRT call routine
RETR	8374	SCRT return routine

Notes

(1) All routines use R3 as program counter, exit with SEP5, and alter registers, X, D, DF, RE, RF, and location M(R2).
(2) READ and READAH exit with R3 pointing back at READAH.
(3) All five TYPE routines exit with R3 pointing at TYPE5.
(4) RO, R1, and R4.1 are altered while storing registers.

TYPE

The TYPE routine has four different entry points. Three of them simply specify different places to fetch the character from: TYPE types from RF.1, TYPE5 types from M(R5) and increments R5, and TYPE6 types from M(R6) and increments R6. TYPE 2 is an entry which results in RF.1 being typed out in hex form as two hex digits. Each 4-bit half is converted to a ASCII hex digit (0-9, A-F) and separately typed out.

Notice that the READ routines are designed to facilitate repeated calls to READAH, while the TYPE routines are designed for repeated calls to TYPE5.

OSTRNG

Another routine, OSTRNG, can be used to output a string of characters. OSTRNG picks up the character string pointed to by R6 and tests each character for zero. The characters should be already encoded in ASCII. If a zero is found (ASCII 'null'), the program terminates and returns to the caller via a SEP R5. If the character is not zero, it is typed out to the terminal.

Tables IX and X include summaries of the functions and calling sequences just described.

Table X - UT71 Register Usage

Register Name	Register Number	Function and Comments
SP	R2	Stack pointer. UT71 uses R2 = 8CFF
PC	R3	Program counter for UT71
CALL	R4	Call routine pointer
RETN	R5	Return routine pointer
LINK	R6	Subroutine data link
ASL	RD	Assembled into by READAH (input hex digits)
AUX	RE	RE.1 holds echo bit. RE.0 is used by all READ and TYPE routines and by OSTRNG and CKHEX.
CHAR	RF	RF.1 holds input/output ASCII character. RF.0 is used by all READ and TYPE routines and by OSTRNG and CKHEX.

Examples of READ and TYPE Usage

The following examples should help clarify how to use the UT71 READ and TYPE subroutines. Most examples use the standard subroutine linkage which requires that R2 point at a free RAM location.

READ Routine

This sample program will read four ASCII-hex characters into register RD translating them from ASCII to hex in the process. Reading will terminate when a carriage return is entered. Entry of a non-hex digit other than a carriage return will cause a branch to an error routine written previously by the user. This sample program uses the standard Subroutine Call and Return Technique (SCRT).

```
READAH EQU 813BH

LOOP  SEP R4;          ..Call the hex
      DC (READAH)      ..read program

      BDF LOOP         ..As long as ASCII hex
                       ..digits are entered,
                       ..read and shift in.
                       ..Fall through is not hex
                       ..character.

      GHI RF           ..See what character was
                       ..last entered.

      XRI ODH          ..Was it carriage return?

      BNZ ERROR        ..If not, BR to error.
                       ..Characters entered are
                       ..now in RD.
```

The READ routine (at 813EH) could be used similarly to enter characters; however, READ only enters them one at a time into RF.1 writing over the previous entry. An alternative technique is to use R5 as the main program counter (since all READ and TYPE routines terminate with a SEP R5) and call the program with a SEP R3 (since all READ and TYPE routines use R3 as their program counter). The following example illustrates this technique.

TYPE Routines

Example 1 (TYPE5). This program outputs a single character using the TYPE5 routine. It uses R5 as the program counter.

```
LDI 81H      ..Set R3 to TYPE5 routine
PHI R3
LDI OAOH
PLO R3
LDI OFFH     ..Set R2 to free RAM location
             ..3FFFH
PLO R2
LDI 3FH
PHI R2
SEP R3;      ..Call type
DC 'R'       ..An "R" will be typed
YY           ..Next instruction
```

Example 2 (TYPE6). This program outputs a character using the TYPE6 routine. When called using the Standard Call and Return Technique, this routine is particularly useful for typing an immediate byte. After typing the byte at M(R6) (which is pointing to the byte immediately following the call) a return is made to the caller past the typed byte.

```
SEP R4;      ..Branch to the call routine
DC 81A2H     ..Address to TYPE6
DC '?'       ..Byte to be typed out
YY           ..Next instruction
```

Example 3 (TYPE and TYPE2). The TYPE and TYPE2 routines pick up the byte in RF.1 for typing. TYPE simply outputs the character, whereas TYPE2 considers RF.1 a hex digit pair which it encodes in ASCII before typing. This example types out the hex digits 'D5' and uses Standard Call and Return Technique.

```
LID OD5H     ..Load hex digits D5
PHI RF       ..Into RF.1
SEP R4       ..Call TYPE2
DC 81AEH
YY           ..Next instruction
```

Note that all type routines, except TYPE2, expect the character they pick up to be already encoded in ASCII.

Example 4 (OSTRNG). An entire message can be typed by using the OSTRNG routine. The ASCII bytes pointed to by R6 will be typed. When a '00' byte is detected, OSTRNG returns to the caller. This example will output the string.

```
RCA COSMAC
MICROPROCESSOR
```

The Standard Call and Return Technique should be used.

```
OSTRNG EQU 83FOH
   .
   .
SEP R4;
DC (OSTRNG)              ..Call OSTRNG
DC 'RCA COSMAC'          ..1st Line
DC ODOAH                 ..(CR)(LF)
DC 'MICROPROCESSOR'      ..2nd Line
DC OOH                   ..End of Text
```

10. Additional Monitor Routines

ASCII to Hex Conversion (CKHEX)

The ASCII to hex conversion routine, CKHEX, examines the ASCII character in RF.1. If this character is not a hex digit, CKHEX returns to the user (via SEP R5) with DF=0. If the character is hex, CKHEX returns with RE.0 = hex digit, DF=1 and with the digit shifted into the least significant 4 bits of register RD. CKHEX uses the registers described above and, as with the other routines, is most readily handled via the Standard Call and Return Techniques. CKHEX is located at 83FCH.

Initialization Routines (INIT1 and INIT2)

Two routines are provided, INIT1 and INIT2, to initialize CPU registers for the Standard Call and Return Technique. These routines set up registers as follows:

R2=R(X)	-pointing to 8CFFH
R3	-will become the program counter on return
R4	-pointing to the CALL routine in UT71
R5	-pointing to the RETURN routine in UT71

The only difference between INIT1 and INIT2 is the location to which they return. INIT1 returns to location 0005 with P = 3; INIT2 simply returns by setting P = 3 and assumes that the user has already set R3 pointing to the correct return point. These programs are intended as a convenience to free the user from generating the overhead code required by the standard subroutine technique. They may also be used as an integral part of custom support programs running on the MS2000. Their absolute addresses are INIT1 EQU 83F3H and INIT2 EQU 83F6H

The INIT routines should be used to set up R4 and R5. Following are examples of the use of these programs:

Example 1 (INIT1): INIT1 EQU 83F3H

Address	Code	Mnemonics	Comments
0000	71	DIS, 0	..Disable interrupts
0001	00		
0002	C0	LBR INIT1	..Initialize registers
0003	83		
0004	F3		
0005	-	(USRPGM)-	..User program starts ..here; P=3, X=2

Example 2 (INIT2): INIT2 EQU 83F6H

Address	Code	Mnemonics	Comments
0000	71	DIS,0	..Disable interrupts
0001	00		
0002	F8	LID A.1 (START)	..Set R3 to return ..Point
0003	00		
0004	B3	PHI R3	
0005	F8	LDI A.0 (START)	
0006	50		
0007	A3	PLO R3	
0008	C0	LBR INIT2	..Call INIT2
0009	83		
000A	F6		
.			
.			
0050	-	START-	..User program starts ..here; P=3, X=2

Restarting UT71 (GOUT71)

A means is provided to automatically transfer control back to UT71 from a user program. An entry point routine, GOUT71, is provided for this purpose. When entered via this routine, UT71 will restart and issue a * prompt to the terminal. A long branch to GOUT71 at location 83F9H will cause this transfer.

Line Printer Interfacing (LINEPR)

The utility routine LINEPR located at 850EH is supplied for line printer interfacing. It will output the byte in RF.1 to a line printer port. Line feeds are suppressed, but carriage returns are replaced with a line feed-carriage return pair. Return is made with DF=1, unless the character in RF.1 is an ASCII 'DC3' (end-of-file marker). In that case, the DC3 is not output, and DF=0 on return. This routine should be called with the Standard Call and Return Technique.

Disk Routines

The loader is a routine that loads memory by doing track reads. It can load the 12-kilobyte MicroDOS operating system in approximately one second. Data is transferred to memory by DMA starting at address 9000H to BFFFH.

The loader resides in memory starting at 8400H. It requires a RAM area to set up a buffer containing the bytes to be output to the disk controller and to store the resulting status information. In addition, a stack area is required for operation. RAM area between 8F00H and 8FFFH is used for this purpose.

To load the operating system, first place a diskette containing MicroDOS into drive 0. Then type L after the * prompt. After the operating system is loaded, it will print a header followed by a > prompt, indicating that it is ready to accept MicroDOS commands.

If the user wants to go back to the monitor, he can use the U utility command and enter $U 8000. The monitor will issued the * prompt and wait for monitor commands. The user can go back to the operating system by entering P 9000.

If the user did not initially insert a diskette in drive 0, or if the data that was loaded into memory was not an operating system, the following will be printed:

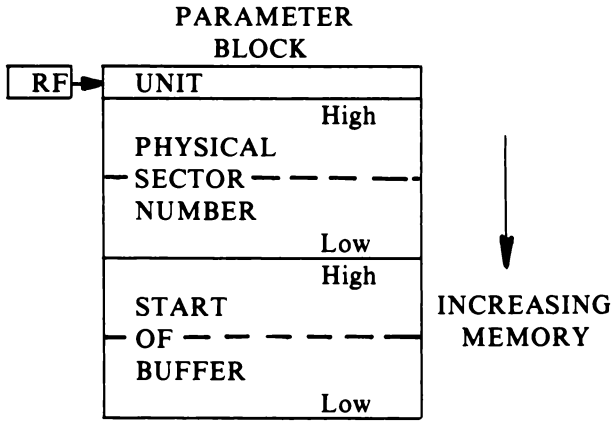


Fig. 21 - Conditions for calling SEEK, READ, and WRITE routines.

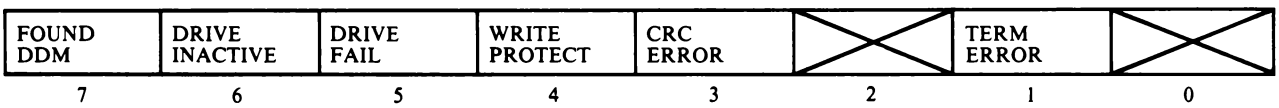


Fig. 22 - RD.0 Status byte showing arrangement of bits after a driver function is finished.

```
INIT1   EQU  83F3H
SEEK    EQU  87F6H
READ    EQU  87F9H
WRITE   EQU  87FCH

        DIS           ..DISABLE
                      INTERRUPTS
        DC 0
        LBR INIT1     ..INITIALIZE
                      REGISTERS

        A.1(PARM)-
         -> RF.1      ..POINT AT PARA-
                      METER BLOCK
        A.0(PARM)-
         -> RF.0

START   CALL SEEK     ..SEEK TO TRACK 0
        CALL READ     ..READ PSN 0 INTO
                      MEMORY
        CALL WRITE..WRITE MEMORY
                      BACK TO PSN 0
        LBR START     ..DO IT AGAIN

        ORG 1000H     ..PARAMETER BLOCK
PARM    DC 00H        ..UNIT 0
        DC 0000H      ..PSN 0
        DC 2000H      ..READ/WRITE
                      BUFFER
```

Fig. 23 - Example demonstrating use of SEK, READ, and WRITE routines.

MICRODOS NOT LOADED

and the monitor will reissue the * prompt.

The monitor also contains the routines SEEK, READ, and WRITE. These routines perform the actual driver functions that link the operating system with the disk drives.

Calls to Driver Routines

The following information is for users who may want to utilize the disk I/O routines in UT71.

The SEEK, READ, and WRITE routines must be called in accordance with the conditions shown in Fig. 21.

After the driver function is finished, RF will remain pointing at the unit byte. RD.0 will contain a status byte showing the result of the operation. Fig. 22 shows the arrangement of the status bits in RD.0.

The example in Fig. 23 demonstrates the use of the SEEK, READ, and WRITE routines in UT71. It is a complete program that will continuously read from and write to PSN 0 on drive 0. Programs written by the user should test the status bits in RD.0 after each call to a disk routine to determine if that function was successfully performed. Recovery from failed functions should be accomlished with retry logic.

Appendix A - Diskette Organization and Structure

Each diskette has 70 tracks with 9 sectors on each track (630 sectors per diskette). However, from MicroDOS's point of view, the diskette is divided up into clusters with 1 sector in each cluster.

The system diskette has two basic configurations, one with a directory and operating system and one with a directory only. These configurations are generated with the SYSGEN command. Because the operating system requires about 4 per cent of the diskette, diskettes with directory only have more disk area for storage of the user's work files.

MicroDOS assumes that a file is a string of bytes. When a file is created, a certain number of clusters is allocated to it. If more space is needed for the data than initially allocated, MicroDOS automatically allocates more space. Once a file has been created by the user, the operating system returns to the system any unused disk cluster so that the next file to be created can use this freed-up space. No cluster can be allocated to two different files.

Diskette Information Format

TRACK 0

Sector 1 = DISK ID

Bytes	0 - 11	Unused
Bytes	12 - 19	Date (8 ASCII characters)
Bytes	20 - 63	User ID (44 ASCII characters
Bytes	64 - 511	Unused

Sector 2 - 9 = DISK DIRECTORY

Every 16 bytes = one file directory entry
Within an entry:

Bytes	0 - 5	First part of filename (6 ASCII characters)
Bytes	6 - 8	Filename extension (3 ASCII characters)
Bytes	10 - 11	Starting Sector Number (in hexadecimal)
Byte	12	Attribute code

TRACK 1

Sector 1 = CLUSTER ALLOCATION TABLE

The first 623 bits indicate the status of the 630 clusters on the disk: 1 = in use, 0 = free. Each cluster has 1 sector in it. Note that there are:

512 bytes/sector	630 clusters/disk
1 sector/cluster	630 sectors/disk
9 clusters/track	70 tracks/disk
512 bytes/cluster	322,560 bytes/disk side

NOTE: Tracks are numbered 0 - 69 (00H-45H)
Sectors are numbered 1 - 9 (01H-09H)
Bytes are numbered beginning at 0
Bit 0 is the LSB on right-most bit in a byte

Start Sector Number (SSN)

The integer portion of the quotient SSN/9 equals the track number, while the remainder +1 indicates the sector within the track. For example, sector 114 is located at sector 7 on track 12.

Non-contiguous files may be broken up into 1 to 57 segments, which may be distributed throughout the disk. A segment my contain 1 to 128 contiguous clusters depending on how much contiguous free space there is at that location on the disk.

The first sector of the first segment of any file is the SSN given in the disk directory. It is called the Retrieval Information Block (RIB) and contains information needed to locate all segments of the file. The file's data starts in the sector following the RIB.

RIB (located by the SSN given in the directory)

Each 24 bits may contain one Segment Descriptor Word. SDW's are of two types:

SDW: (If file takes more than 1 segment

Bits 0 - 15	= PSN where segment starts
Bits 16 - 22	= number of contiguous clusters (minus 1 allocated to this segment)
Bit 15	= 0 since more SDW's follow in this RIB

LAST SDW:

Bits 0 - 14	= total number of sectors actually used in file.
Bit 15	=1 to indicate it is the last SDW.

For binary files the RIB also contains:
Bytes 500-501 = number of bytes in the last sector
Bytes 502-503 = number of sectors to load
504-505 = starting load address in RAM
Bytes 506-507 = entry address for program execution

CRC errors that show up during a disk write and persist after five tries cause a deleted data mark to be placed in the sector and that sector is passed over without losing data. That sector is never used again by MicroDOS.

Free space, however, is determined by the number of unused clusters and does not reflect the unusable sectors with DDM's.

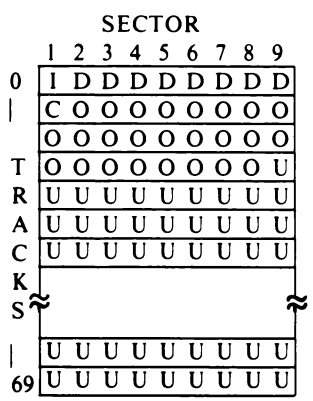
Physical Structure (Decimal PSW)

Number	Letter	Contents
0	I	Disk ID
9	C	Cluster Allocation Table
1 - 8	D	Directory
10 - 34	O	Operating System
35	U	Unused

		Decimal	Hexa-decimal
Sectors per track	9	1 - 9	1 - 9
Tracks per disk	70	0 - 70	0 - 46
Sectors per disk	630	0 - 629	0 - 275
Sectors per cluster	1	-	-
Clusters per disk	630	0 - 629	0 - 275

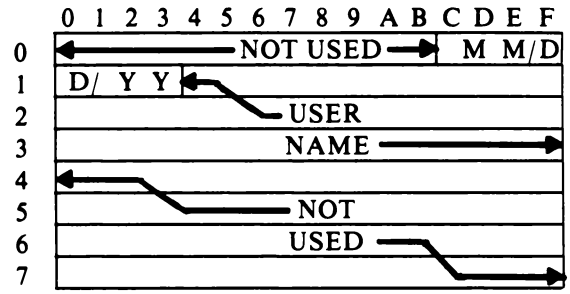
Diskette Structure

Following is a set of diagrams that describe the disk structure of MicroDOS.



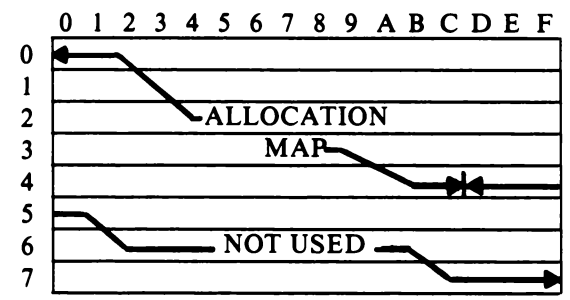
I=Disk Identification Block
D=Directory
C=Cluster Allocation Table
O=Operating System
U=Unused

Disk Identification Block



Byte	Size	Contents
0 - BH	12	Not used
CH - 13H	8	Creation date
14H - 3FH	44	User name
40H - 1FFH	-	Not used

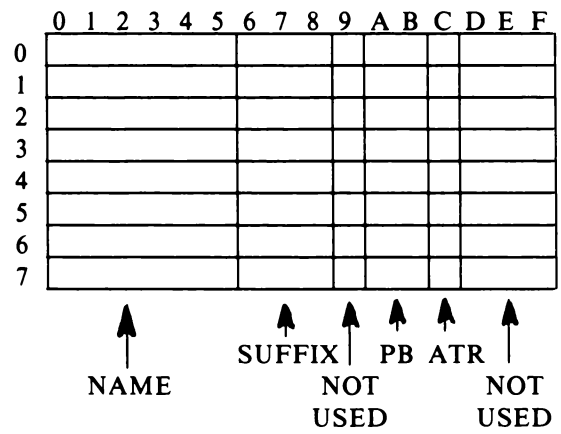
Cluster Allocation Table (CAT)



Offset	Size	Contents
0 - 4DH	78	Cluster Allocation Table
4EH - 1FFH	-	Not used

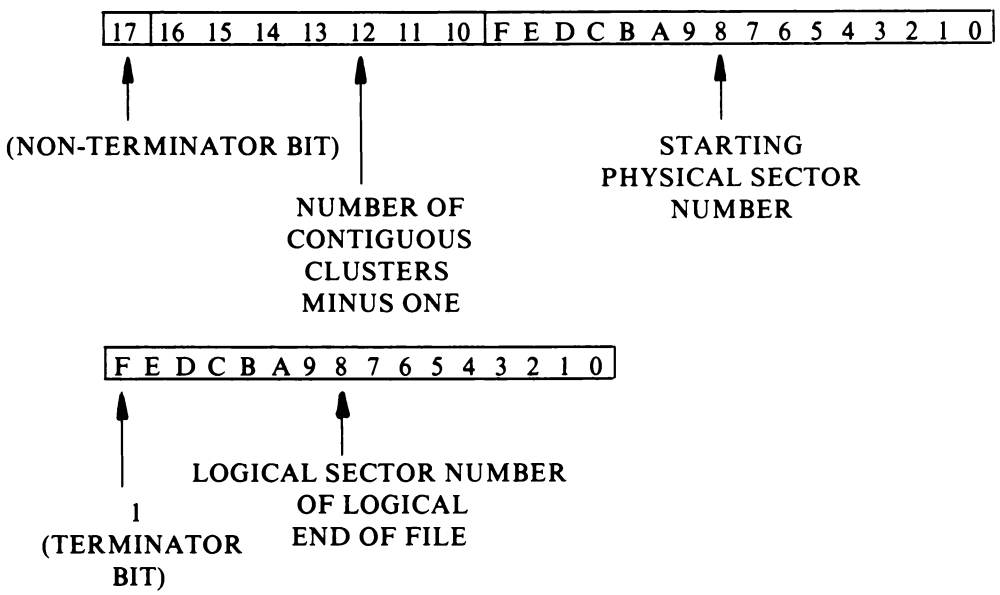
Each byte of the Cluster Allocation Table (CAT) contains 8 bits for 8 clusters of allocation. Byte 4DH must have bit 0 set to "1" because no sector corresponds to this cluster number. All unused bytes have bits set to "1".

Directory Sectors (DIR)



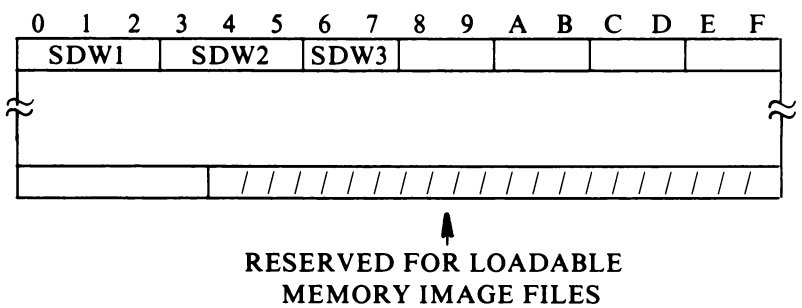
Byte	Size	Contents
0 - 5	6	Filename
6 - 8	3	Suffix
9	1	Not used
AH - BH	2	Physical address of Pointer Block (PB)
CH	1	Attributes
DH - FH	3	Not used

Segment Descriptor Word (SDW)



1 SDW Can span 127 Clusters

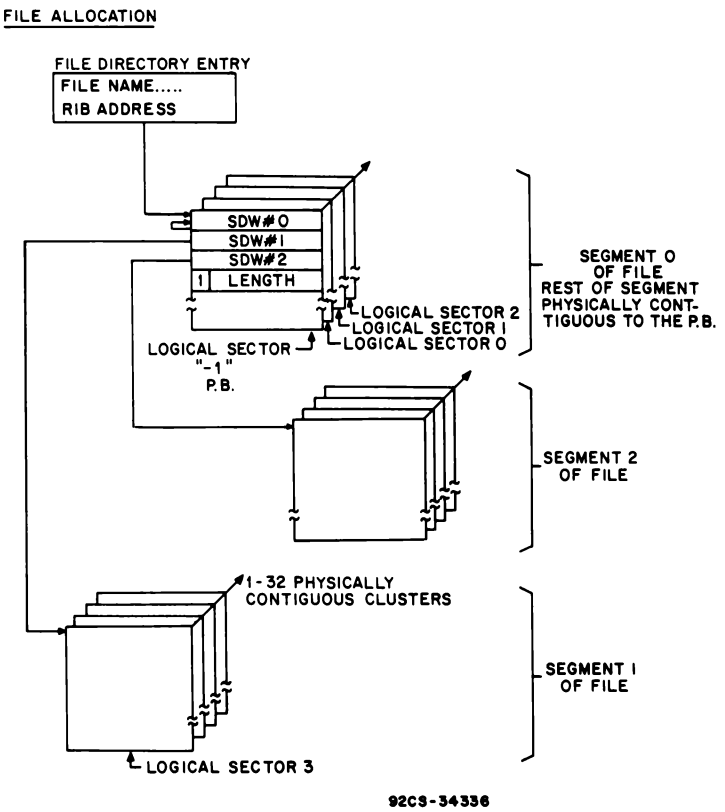
Pointer Block



There can be from 1 to 57 Segment Descriptor Words (SDW) plus a terminator SDW.

Unused SDW's after terminator words are 0 (except for memory image files).

File Allocation



Appendix B
BNF Syntax of Assembler ASM8

The following is a compilation of the full BNF (Backus-Naur Format) description of the assembly language, ASM8. In these descriptions, the symbol "::=" means "is defined as." Where there is choice between alternatives, the symbol "!" is used to separate the choices. Angle brackets "<" and ">" are used to indicate a non-terminal element, i.e., a description of something. A terminal element is written exactly as it would appear when used.

<binary digit> ::= 0!1
<octal digit> ::= <binary digit>!2!3!4!5!6!7
<decimal digit> ::= <octal digit>!8!9
<hexadecimal digit> ::= <decimal digit>!A!B!C!D!E!F
<character> ::= Any printing ASCII character
<character string> ::= <character>!<character string><character>
<break character>::=_
<alphanumeric character>::=<letter>!<decimal digit>!<break character>
<identifier> ::=<letter>!<identifier><alphanumeric character>

Note: An identifier may have no more than nine alphanumeric characters including multiple adjacent break characters.

<space> ::= Δ!<space>Δ
Note: The symbol Δ represents the ASCII space character 20H.

<literal constant> ::= !<character string>!

Note: A literal constant may not contain a quote.
Note: A literal constant is ASCII encoded.

<binary constant> ::= <binary digit>B!<binary digit><binary constant>
<octal constant ::= <octal digit>Q!<octal digit><octal constant>
<decimal constant> ::= <decimal digit>!<decimal digit>D!<decimal digit><decimal constant>
<hexadecimal string> ::= <decimal digit>!<hexadecimal string><hexadecimal digit>
<hexadecimal constant> ::= <hexadecimal string>H
<constant> ::=<binary constant>!<octal constant>!<decimal constant>!<hexadecimal constant>

Note: A constant is truncated to the last two bytes of its hexadecimal equivalent.

<location counter< ::= $
<dummy> ::= [<identifier>]
<least significant byte> ::= A.0(<term>)
<most significant byte> ::= A.1(<term>)
<element> ::= <identifier>!<constant>!<literal constant>!<location counter>!<dummy>!<least significant byte>!<most significant byte>!(<term>)!<element><space>!<space><element>

<factory> ::=<element>!-<element>!+<element>!<factor>*<factor>!<factor>/<factor>!<factor><space>!<space><factor>
<term> ::= <factor>!<term>+<term>!<term>-<term>!<term><space>!<space><term>
<relational operator> ::= .EQ.!.GT.!.LT.!.LE.!.GE.!.NE.
<relation> ::= <term><relational operator><term>!<relation><space>!<space><relation>!.NOT.<relation>!(<logical term>)!<term><logical element> ::= <relation>!<logical element><space>!<logical element>.AND.<logical element>!<space><logical element>
<logical factor> ::= <logical element>!<logical factor><space>!<space><logical factor>!<logical factor>.XOR.<logical factor><logical term> ::= <logical factor>!<logical term><space>!<space><logical term>!<logical term>.OR.<logical term>

Note: No expression (logical element, logical factor, logical term, relation, element, factor, or term) may contain more than twenty elements or more than twenty operators. (+, -, *, /, A.1, A.0, RELATIONAL OPERATOR, .NOT., .AND., .OR., .XOR.)

<first class instruction> ::= IDL!NOP!SEQ!REQ!SAV!MARK!RET!DIS!LDX!LDXA!STXD!IRX!OR!XOR!AND!SHR!SHRC!SHL!SHLC!ADD!

ADC!SD!SDB!SM!SMB!SKP!LSKP!LSZ!LSNZ
!SNF!LSQ!LSNQ!LBNQ!LSIE!LDC!GEC!
STPC!DTC!STM!SCM1!SCM2!SPM1!SPM2!
ETQ!XIE!XID!CIE!CID!BCI!BXI!DADD!
DADC!DSM!DSMB!DSAV

<second class instructor> ::= SEP!SEX!LDN!L
DA!STR!INC!DEC!GLO!PLO!GHI!PHI!RLXA!
RSXD!RNX!SRET

<third class instructor> ::= LDI!ORI!XRI!ANI!
ADI!ADCI!SDI!SDBI!SMI!SMBI!BR!NBR!BZ!
NBZ!BDF!BPZ!BGE!BNF!BM!BL!BQ!BNQ!OUT!
INP!LBR!LBZ!LBNZ!LBDF!LBNF!LBQ!LBNQ!
NLBR!DADI!DACI!DSMI!DSBI

<fourth class instruction> ::= RLDI!SCAL!DBNZ

<register> ::= R<hexadecimal digit>!<term>!
<register><space>!<space><register><immediate
operand> ::= <term>!<literal constant><operand
string> ::= <register>!<immediate operand>!
<operand string><space>!<space><operand
string>!<operand string>,<operand string>

Note: An operand string may not have more than 76 characters, including those inserted by the assembler.

<immediate keyword> ::= IDLE!GOTO!NOGOTO!
SKIP!RETURN!DISABLE!POP!PUSH!SAVE!
GOSTATE!CALL!EXIT<branch keyword> ::= 0!Q
!&=0!DF!PZ!GE!EF1!EF2!EF3!EF4!NQ!&>0!>
NDF!MINUS!LESS!NEF1!NEF2!NEF3!NEF4

<substitution> ::= IF<space><branch keyword>
<space>GOTO!<immediate keyword><load part>
::= @!@!!@<register>!@<register>!!@"<register>
!@(<character string>)!<term>!<register>.0!
<register>.1

Note: the above character string may not contain parentheses.

<operator> ::= +!-!-+!+"!-"!-+"!.AND.!.OR.!.XOR.
<object> ::= @!@"<register>!<term>

<manipulation part> ::= <operator>
<object>!/2!*2!/2"!*2"
<arrow> ::= →>
<storage part> ::= <arrow><register>.0!<arrow>
<register>.1!<arrow>@<register>!<arrow>
@-!<arrow>@-"<register>!<arrow>@-(<character
string>)<D-sequence statement> ::= <load
part>!<manipulation part>!<storage part>!<load
part><manipulation part>!!<load part><storage
part>!<manipulation part><storage part>
!<load part><manipulation part><storage part>
<level II statement> ::= <substitution>!<D-sequence
statement>

ote: A level II statement may not contain more than thirty-nine characters.

<executable statement> ::= <first class statement>
!<second class instruction><space><register>
!<third class instruction><space><immediate
operand>
!<fourth class instruction><space><register>,
<immediate operand>
!<level II statement>
!<executable statement><space>!<space><ex-
ecutable statement>
<statement set> ::= <executable statement>!<state-
ment set><space>!<space><statement set>!<state-
ment set>;<statement set>

Note: A statement set may have no more than ten executable statements.

<macro name> ::= <identifier>
<macro call statement> ::= <macro name>
!<macro name><space><operand string>
<label> ::= <identifier>
<comment> ::= ..<character string>
<line beginning> ::= <space>!<label><space>
<line ending>::=carriage return>!<space>>line
ending>!<comment><line ending>
<executable line> ::= <line beginning><statement
set><line ending>!<line beginning><macro call
statement><line ending>

<end statement> ::= END!END<space><label>
<eject statement> ::= EJECT
<nolist statement> ::= NOLIST
<list statement> ::= LIST
<macro statement> ::= MACRO
<endm statement> ::= ENDM
<non-terminal major statement> ::= <eject state-
ment>!<list statement>!<nolist statement>
<non-terminal major line> ::= <line beginning>
<non-terminal major
statement>
<line ending>
<non-terminal line>::=<executable line>!<non-
terminal major line>
<equate statement> ::= <label><space>EQU
<space><term>!<label><space>EQU<space>R
<hexadecimal digit>
<constant declaration> ::= <line beginning>DC
<space><operand string>
<storage declaration> ::= <label><space>DS
<space><term>
<org statement>::= <label><space>ORG
<space><term>
<page statement> ::= <label><space>PAGE
<minor statement> ::= <equate statement>
!<constant declaration>!<storage declaration>
!<org statement>!<page statement>

<minor line> ::= <minor statement>
<line ending>

<end line> ::= <line beginning><end statement>
<line ending>
<macro line> ::= <line beginning>
<macro statement><line ending>
<endm line> ::= <line beginning>
<endm statement><line ending>
<if statement> ::= IF<space><lobical term>
<else statement> ::= ELSE
<endif statement> ::= ENDIF
<value> ::= <constant>!<value><space>
!<space><value>

Note: Value will be truncated to 1 byte

<increment list> ::= <value>,<value>,<value>
<replacement list> ::= <operand string>
<increment marker> ::= =!<space>=
<replacement marker>::= :!<space>:
<do statement> ::= DO<space><dummy>
<increment marker><increment list>
!DO<space><dummy><replacement marker>
<replacement list>
<endd statement> ::= ENDD
<go statement> ::= GO<space><label>
<exitm statement> ::= EXITM
<if line> ::= <line beginning><if statement>
<line ending>
<else line> ::= <line beginning><else statement>
<line ending>
<endif line> ::= <line beginning>
<endif statement><line ending>
<do line> ::= <line beginning><do statement>
<line ending>
<endd line> ::= <line beginning>
<endd statement><line ending>
<go line> ::= <line beginning><go statement>
<line ending>
<exitm line> ::= <line beginning>
<exitm statement><line ending>

Note: No line may contain more than 80 characters

<line block> ::= <non-terminal line>!<if block>
!<do block>!<go line>!<line block><line block>
<if block> ::= <if line><else line><endif line>
!<if line><line block><else line><endif line>
!<if line><else line><line block><endif line>
!<if line><line block><else line><line block>
<endif line>
<do block> ::= <do line><line block>
<endd line>
<dummy list> ::= <dummy>!<dummy list>
<space>!<space><dummy list>
!<dummy list>,<dummy list>
<macro definition> ::= <line beginning>
<macro name><space><dummy list><line
ending>
<macro block> ::= <line block>!<exitm line>
!<macro block><macro block>
<macro> ::= <macro line><macro definition>
<macro block><endm line>
<macro library> ::= <macro>!<macro library>
<macro library>

<source code> ::= <line block>!<line block>
<macro library>
!<source code><end line>

Note: The cumulative size of all macros must not exceed twelve kilobites.

Note: The substitution list may not exceed forty-three characters in length.

Note: If there are more than six errors on a line, or more than one hundred and twenty-eight errors in a program, the assembler may not be able to continue processing.

Appendix C
MS2000 Memory Test

The MicroDOS System Diskette includes a file, MEMTST.CM, that contains a memory test program for the 60 kilobytes of RAM. The user can call up this program at any time to verify that the RAM is functional. It should be noted, however, that this test will write over any program that is located in the RAM.

The memory test checks RAM from location 0000 to 7FFF and from 9000 to FFFF. In this test a "March" pattern is executed with various combinations of the 8-bit data word. The test takes ten minutes to complete, then auto-loads MicroDOS.

Test Procedure

The procedure for the memory test is as follows:

1. Type MEMTST (CR)
2. System will type out

   MEMORY TEST STARTED
3. If no failures are encountered, after ten minutes the System will type out

   MEMORY TEST COMPLETED
4. The program will then load MicroDOS.
5. If any failures are encountered, the System will type out the address of the page on which the failure occurred and then skip to the next page of memory to continue testing. After all memory is checked, the System will type out to the screen

   MEMORY TEST COMPLETED

   and the program will then load MicroDOS.

Board Repair

For information on the repair of faulty boards, contact:

Customer Service, Tel. 800-722-0094
RCA Corporation
New Holland Ave.
Lancaster, PA 17604

Example

Following is an example of a display resulting from the MicroDisk memory test.

MEMORY TEST STARTED
ERROR AT ADDRESS 46XX
ERROR AT ADDRESS 46XX
ERROR AT ADDRESS 94XX
ERROR AT ADDRESS 94XX
ERROR AT ADDRESS CFXX
ERROR AT ADDRESS CFXX

MEMORY TEST COMPLETED

This example indicates that there were errors in three pages at address locations 46XX, 94XX, and CFXX. Note the redundant reporting as a result of repeated testing with different patterns. If a RAM package is completely nonfunctional, missing, purposely disabled, or has been replaced with a ROM, there will be a long stream of error reports. If a single bit is faulty, there will be fewer reports, depending on how many patterns fail.

If a failure is detected, first determine which memory board is at fault. There are two memory boards, both types CDP18S628. They differ in their address locations and in that one has 32 kilobytes of RAM and the other 2 kilobytes of ROM followed by 30 kilobytes of RAM.

Error Address	Faulty Board
OOXX through 7FXX	32-kB RAM
88XX through FFXX	ROM/RAM

To diagnose the faulty board to the chip level, refer to MB-628, **"RCA CMOS Microboard Memories,"** for details of the physical address map. The memory packages are socketed, so that replacement or swapping is easy. Before anything else is done, however, check the linking of the board to see that no changes have been made. Some users may, for example, replace RAM with ROM in order to test software that has been developed and placed in ROM, and may fail to replace the RAMs or to properly relink the board.

Appendix D
Error Messages

1. MicroDOS Error Messages

0. ERR=XX

Where XX = 00 - Tried to open an already opened or reserved file. Make sure that the open parameter and unit number are initialized correctly in the IOCB.

Where XX = 01 - DDM could not be written.

1. CRC ERR—X DR Y-PSN Z

X is the location in the operating system or user program that caused the CRC error.

Y is the drive number.

Z is the physical sector number.

If the CRC happened on a WRITE, an attempt to write five times is tried before a DELETED DATA MARK (DDM) is written in that sector, and the data is attempted to be written onto the next logical sector.

If the CRC happened on a READ and attempted to be reread five times, the data will be passed back to the program for processing.

Sectors with DDM will be skipped on a READ function. No error will be printed.

2. DIR FULL

No more room exists in the directory for a new entry. A new diskette must be used.

3. DISK FULL

No more room exists on the disk for writing. A new disk must be used or data deleted from the current disk. When this error message is generated by any program except ASM8 or EDIT, the incompleted file should be deleted.

4. ILLEGAL DR. #

A number other than 0 or 1 was used for the logical disk number.

5. NOT USED

6. X DOES NOT EXIST

X is a filename.

7. ILLEGAL F.N.

The filename typed is not a valid filename.

8. <FILENAME> DUP. F.N.

The filename typed is a duplicate filename.

9. NO SUCH DV

The chosen device is not part of the current system. A command that would cause this error message is copy SYX,#DRUM where #DRUM is not a valid system device.

10. INVALID DV

The device chosen cannot be used in this situation.

11. COMMAND SYNTAX ERR

An error occurred in syntactically analyzing the command line. Retype the correct command.

12. NOT USED

13. OPTION CONFLICT

There was a conflict in the option selections.

14. INVALID TYPE OF OB FILE

The file to be loaded was not of the correct file type.

15. INVALID LOAD ADDRESS

The load address is out of range of the current machine.

16. NOT USED

17. INVALID RIB

The linkage structure of the disk has been destroyed. Generally this message means that a non-MicroDOS diskette is assumed to be a MicroDOS diskette.

18. INVALID EXEC ADR

This message means that the address is not part of the loaded file.

19. INVALID FILE TYPE

The type of file is not acceptable for use.

20. LOG SECT NO. OUT OF RANGE

The logical sector number was greater than the maximum value or was greater than the end of file.

21. NOT USED

22. <FILENAME> F.N. NOT FOUND

The filename was not found in the specified directory. DIR can be used to list out the filenames.

23. <FILENAME> FILE IS DELETE PROTECTED

<FILENAME> has the delete-protected attribute set.

If the file is to be deleted, remove the protection with the RENAME command and re-execute the DEL function.

24. CONFLICTING FILE TYPES
The file type being read from or written to did not conform to the use.

25. INVALID DATA TRANSFER TYPE
The file type of the file did not conform to the device it was being dumped to.

26. FILENAME IS WRITE PROTECTED
The filename cannot be written to because it has the write-protection attribute set. This error can be corrected by using the RENAME command.

27. NOT USED

28. NO RAM AT XXXX
When a file is being loaded, the RAM area does not exist for the load address.

29. FORMAT ERROR
The ASCII-HEX file does not conform to the correct format.

30. DV NOT READY
The selected device is not ready to accept or send data. This message is issued before the transfer begins.

31. XX DR INACTIVE
This message means that the disk drive is not turned on.

32. XX DR FAIL
The disk drive does not have a diskette properly inserted in the unit.

33. XX LOG. EOF
The program requested more information from the disk file than the disk file had. Usually, no DC3 was present on the input file.

34. XX FILE NOT OPENED
The file being accessed in unit XX was not properly opened before it was used.

35. TRM ERR—DR Y-PSN Z
Termination error occurred at Y drive number and Z physical sector number.

36. DDM ERR—DR Y-PSN Z
Could not write out a DDM at Y drive number and Z physical sector number.

80H. Same as 1

81H. Same as 36

82H. Same as 35

C0H. Same as 0

C1H. Same as 0

C8H. Same as 34

C9H. Same as 33

2. Utility Program UT71 Error Message

ERROR – This message is the result of an error in syntax during the entry of a command to the monitor.

The following error messages are from the monitor self-test routine.

UART BAD – Status byte read back from the UART was not C0H

PROM BAD – The contents of the monitor, after EXCLUSIVE OR'ing every byte, did not match a reference value.

RAM BAD – Memory from 8800 to 8FFF was not able to pass a write to and read back test.

The following error message is from the monitor operating system loader routine.

MICRODOS NOT LOADED – Results if no disk is in drive 0, if a problem occurred during disk I/O, or if the data that was loaded was not MicroDOS.

3. EDIT Error Messages

Message	Meaning
LINE TOO LONG	A line that EDIT is attempting to TYPE has more than 78 characters.
BAD COMMAND?? "XXX..X$"	EDIT has found an invalid command in a command string. XXX...X is that part of the string not executed.
<BELL>	Filled work space warning. Delete part of the command before ending the command.
MEMORY FULL "XXX..X$"	EDIT ran out of work space during an execution. XXX..X is the unprocessed part of the command string.
CAN'T SAVE	There is not enough room in the SAVE area.
CAN'T FIND "text"	The specified character sequence was not found between the pointer's previous position and the end of the buffer.
<XX> IS WRITE PROTECTED	The disk unit selected (XX) for output is write protected. The command string is aborted. No lines are written or lost.

<XX> DR FAIL — The disk unit selected for output is not ready. The command string is aborted. No lines are written or lost.

ITERATION STACK FAULT — EDIT ran out of stack space during execution of a command string. May indicate improperly paired brackets in the string.

EOF — A line containing an end-of-file mark (DC3) has been read. The DC3 is stored in the buffer and further appends from the current file are ignored.

DISK FULL SET UP CONTINUATION FILE WRITE? — Output disk full. Replace disk and enter continuation file name after the query WRITE?

Appendix E—
Sample Program Illustrating User Functions

```
!M
0000 ;              0001
0000 ;              0002  ..USER FUNCTION EXAMPLE - COPY A FILE TO ANOTHER FILE.
0000 ;              0003  ..THE FOLLOWING INFORMATION IS A DEFINITION FOR THE PROGRAM:
0000 ;              0004  ..
0000 ;              0005  ..NO_ERRORS:  BOOLEAN = TRUE ;..NOT_EOF:  BOOLEAN = TRUE ;
0000 ;              0006  ..
0000 ;              0007  ..
0000 ;              0008  ..
0000 ;              0009  ..BEGIN USER EXAMPLE
0000 ;              0010  ..        OPEN INPUT FILE ;
0000 ;              0011  ..        OPEN OUTPUT FILE ;
0000 ;              0012  ..        WHILE NOT_EOF AND NO_ERRORS
0000 ;              0013  ..                DO READ SECTOR ;
0000 ;              0014  ..                WRITE SECTOR TO FILE ;
0000 ;              0015  ..                REPEAT;
0000 ;              0016  ..        CLOSE INPUT_FILE ;
0000 ;              0017  ..        CLOSE OUTPUT_FILE ;
0000 ;              0018  ..END USER EXAMPLE
0000 ;              0019  ..
0000 ;              0020  ..
0000 ;              0021  ..
0000 ;              0022  ..START OF CODE
0000 F802B3;        0023  A.1(START)->R3.1
0003 F8F3A3;        0024  A.0(START)->R3.0
0006 D3;            0025  SEP R3
0007 ;              0026  ..
0007 ;              0027  ..
0007 ;              0028        PROGRAM EQUATES
0007 ;              0029
0007 ;              0030  ..    USER FUNCTION EQUATES
0007 ;              0031  ..
0007 ;              0032 CREAD EQU 012H             ..CONSOLE READ
0007 ;              0033 TYPE EQU 014H              ..CONSOLE TYPE
0007 ;              0034 SRNAM EQU 024H             ..SEARCH FOR A FILENAME
0007 ;              0035 CDERR EQU 028H             ..PRINT ERROR MESSAGE
0007 ;              0036 OPEN EQU 0                 ..OPEN A FILE
0007 ;              0037 GETSEC EQU 6               ..GET A SECTOR FROM THE OPENED FILE
0007 ;              0038 PUTSEC EQU 010H            ..PUT A SECTOR TO THE OPENED FILE
0007 ;              0039 CLOSE EQU 2                ..CLOSE THE FILE
0007 ;              0040 CDENT EQU 01EH             ..RETURN TO THE OPERATING SYSTEM
0007 ;              0041 UCALL EQU 0B453H           ..ADDRESS FOR THE UCALL ROUTINE
0007 ;              0042 SRNERR EQU 0B452H          ..ADDRESS FOR SRNAM ERROR BYTE
0007 ;              0043  ..
0007 ;              0044
0007 ;              0045  ..     IOCB OFFSET EQUATES
0007 ;              0046  ..
0007 ;              0047  ..
0007 ;              0048 OPENPR EQU 0               ..OPEN PARAMETER
0007 ;              0049 STATUS EQU 1               ..IOCB STATUS BYTE
0007 ;              0050 STARTB EQU 5               ..START OF SECTOR BUFFER
0007 ;              0051 ENDBUF EQU 7               ..END OF SECTOR BUFFER
0007 ;              0052 WRITEP EQU 9               ..WRITE PARAMETER
0007 ;              0053 UNITNO EQU 11              ..UNIT NUMBER
0007 ;              0054 FILEDF EQU 24              ..FILE DEFINITION
0007 ;              0055 DEVICE EQU 31              ..DEVICE MNUMONIC
0007 ;              0056 SPACE EQU 020H              ..BLANK CHARACTER
0007 ;              0057  ..
0007 ;              0058  ..
0007 ;              0059  ..      BUFFER AREAS
0007 ;              0060  ..
0007 ;              0061 IOCBR
0007 B1;            0062          DC 0B1H           ..OPEN PARAMETER
0008 ;              0063          DS  4
000C 004F;          0064           DC (INPBUF)     ..START SECTOR BUFFER
```

```
000E 024E;          0065          DC (INPBUF+511)..END SECTOR BUFFER
0010 ;              0066         DS  27
002B ;              0067 IOCBW1
002B 7A;            0068         DC 07AH          ..OPEN PARAMETER
002C ;              0069         DS  4
0030 004F;          0070          DC (INPBUF)     ..START OF SECTOR BUFFER
0032 024E;          0071          DC (INPBUF+511)..END SECTOR BUFFER
0034 ;              0072         DS  27
004F ;              0073 INPBUF DS  512      ..SECTOR BUFFER
024F ;              0074 LINEBF DS  80       ..CONSOLE INPUT BUFFER
029F ;              0075 PACKET DS  4        ..SRNAM PACKET
02A3 ;              0076  ..
02A3 ;              0077  ..
02A3 ;              0078  ..      PROGRAM VARIABLES / CONSTANTS
02A3 ;              0079  ..
02A3 ;              0080  ..
02A3 ;              0081 NOTEOF
02A3 00;            0082         DC 000H          ..END OF FILE FLAG
02A4 ;              0083 ERRFLG
02A4 00;            0084         DC 000H          ..ERROR FLAG
02A5 ;              0085 IEOF EQU 0C9H             ..END OF FILE ERROR NUMBER
02A5 ;              0086 IOCBRG EQU 15             ..REGISTER USED TO POINT TO IOCB
02A5 ;              0087 IOCBPT EQU 12             ..DREGISTER USED TO POINT TO IOCB
02A5 ;              0088 TMPRG1 EQU 14             ..TEMPORARY REGISTER USED BY ROUTINES
02A5 ;              0089 TMPRG2 EQU 13             ..TEMPORARY REGISTER USED BY ROUTINES
02A5 ;              0090 TMPRG3 EQU 11             ..TEMPORARY REGISTER USED BY ROUTINES
02A5 ;              0091 INPMSG
02A5 494E50555420;  0092         DC 'INPUT FILENAME TO BE READ'
02AB 46494C454E41;
02B1 4D4520544F20;
02B7 424520524541;
02BD 44;
02BE 3E;            0093         DC '>'
02BF 00;            0094         DC 000H
02C0 ;              0095 WRTMG1
02C0 494E50555420;  0096         DC 'INPUT WRITE FILENAME>'
02C6 575249544520;
02CC 46494C454E41;
02D2 4D453E;
02D5 00;            0097         DC 000H
02D6 ;              0098 RETYPE
02D6 46494C454E41;  0099         DC 'FILENAME ERROR'
02DC 4D4520455252;
02E2 4F52;
02E4 0D0A;          0100         DC 0D0AH
02E6 524554595045;  0101           DC 'RETYPE NAME>'
02EC 204E414D453E;
02F2 00;            0102         DC 000H
02F3 ;              0103  ..
02F3 ;              0104  ..
02F3 ;              0105  ..
02F3 ;              0106  ..    THIS IS THE MAIN LOOP OF THE PROGRAM
02F3 ;              0107  ..
02F3 ;              0108  ..
02F3 ;              0109  ..
02F3 D4B453;        0110 START  CALL UCALL
02F6 14;            0111         DC TYPE
02F7 02A5;          0112         DC (INPMSG)
02F9 D40348;        0113         CALL OPENR
02FC D4B453;        0114         CALL UCALL
02FF 14;            0115         DC TYPE
0300 02C0;          0116         DC (WRTMG1)
0302 D40387;        0117         CALL OPENW
0305 F802BE;        0118 CP10   A.1(NOTEOF)->TMPRG1.1    ..TEST IF EOF FLAG OR DISK ERROR
0308 F8A3AE;        0119         A.0(NOTEOF)->TMPRG1.0
030B EE;            0120         SEX TMPRG1
030C 4EF1;          0121         @TMPRG1!.OR.@
030E CA0332;        0122         LBNZ CP20                ..BRANCH IF EOF OR ERROR
0311 D4B453;        0123         CALL UCALL
0314 06;            0124         DC GETSEC
0315 0007;          0125         DC (IOCBR)..READ ONE SECTOR
0317 D40419;        0126         CALL CKRERR
031A F802BE;        0127         A.1(NOTEOF)->TMPRG1.1    ..CHECK FOR EOF OR DISK ERROR
031D F8A3AE;        0128         A.0(NOTEOF)->TMPRG1.0
0320 EE;            0129         SEX TMPRG1
0321 4EF1;          0130         @TMPRG1!.OR.@
0323 CA0332;        0131         LBNZ CP20                ..BRANCH IF ERROR ON READ
0326 D4B453;        0132         CALL UCALL
0329 10;            0133         DC PUTSEC
032A 002B;          0134         DC (IOCBW1)..WRITE SECTOR TO FILE #1
032C D40443;        0135         CALL CKW1ER
032F C00305;        0136         LBR CP10
```

```
0332 D4B453,        0137 CP20   CALL UCALL
0335 02;            0138         DC CLOSE
0336 0007;          0139         DC (IOCBR)..CLOSE OUT FILES
0338 D40419;        0140         CALL CKRERR
033B D4B453;        0141         CALL UCALL
033E 02;            0142         DC CLOSE
033F 002B;          0143         DC (IOCBW1)
0341 D40443;        0144         CALL CKW1ER
0344 D4B453;        0145         CALL UCALL
0347 1E;            0146         DC CDENT
0348 ;              0147  ..
0348 ;              0148  ..
0348 ;              0149  ..
0348 ;              0150  ..    OPEN SUBROUTINE
0348 ;              0151  .
0348 ;              0152  ..
0348 D403C6;        0153 OPENR  CALL IOCBIN
034B 0007;          0154         DC (IOCBR)     ..INITIALIZE IOCB
034D D4B453;        0155         CALL UCALL
0350 12;            0156         DC CREAD
0351 024F;          0157         DC (LINEBF)
0353 50;            0158         DC 80..INPUT FILENAME
0354 D4B453;        0159         CALL UCALL
0357 24;            0160         DC SRNAM
0358 029F;          0161         DC (PACKET)..PUT FILENAME INTO IOCB
035A F8B4BE;        0162         A.1(SRNERR)->TMPRG1.1    ..TEST FOR ERROR
035D F852AE;        0163         A.0(SRNERR)->TMPRG1.0
0360 0E;            0164         @TMPRG1
0361 C2036D;        0165         LBZ OPRT15                ..BRANCH IF NO ERROR
0364 D4B453;        0166 OPRT12 CALL UCALL
0367 14;            0167         DC TYPE
0368 02D6;          0168         DC (RETYPE)
036A C00348;        0169         LBR OPENR                 ..REDO NAME
036D D4B453;        0170 OPRT15 CALL UCALL
0370 00;            0171         DC OPEN
0371 0007;          0172         DC (IOCBR)..OPEN FILENAME
0373 F800BF;        0173         A.1(IOCBR+1)->IOCBRG.1
0376 F808AF;        0174         A.0(IOCBR+1)->IOCBRG.0
0379 0F;            0175         @IOCBRG
037A C20386;        0176         LBZ OPRT30                ..BRANCH IF NO ERRORS
037D D4B453;        0177         CALL UCALL
0380 28;            0178         DC CDERR
0381 0007;          0179         DC (IOCBR)..OTHERWISE PRINT OUT MESSAGE
0383 C00364;        0180         LBR OPRT12
0386 D5;            0181 OPRT30 EXIT
0387 ;              0182  ..
0387 ;              0183  ..
0387 ;              0184  ..     WRITE OPEN SUBROUTINE
0387 ;              0185  ..
0387 ;              0186  ..
0387 D403C6;        0187 OPENW  CALL IOCBIN
038A 002B;          0188         DC (IOCBW1)      ..INITIALIZE IOCB
038C D4B453;        0189         CALL UCALL
038F 12;            0190         DC CREAD
0390 024F;          0191         DC (LINEBF)
0392 50;            0192         DC 80..INPUT FILENAME
0393 D4B453;        0193         CALL UCALL
0396 24;            0194         DC SRNAM
0397 029F;          0195         DC (PACKET)..PUT FILENAME INTO IOCB
0399 F8B4BE;        0196         A.1(SRNERR)->TMPRG1.1      ..TEST FOR ERROR
039C F852AE;        0197         A.0(SRNERR)->TMPRG1.0
039F 0E;            0198         @TMPRG1
03A0 C203AC;        0199         LBZ OPWT15                  ..BRANCH IF NO ERROR
03A3 D4B453;        0200 OPWT12 CALL UCALL
03A6 14;            0201         DC TYPE
03A7 02D6;          0202         DC (RETYPE)
03A9 C00387;        0203         LBR OPENW
03AC D4B453;        0204 OPWT15 CALL UCALL
03AF 00;            0205         DC OPEN
03B0 002B;          0206         DC (IOCBW1) ..OPEN FILENAME
03B2 F800BF;        0207         A.1(IOCBW1+1)->IOCBRG.1
03B5 F82CAF;        0208         A.0(IOCBW1+1)->IOCBRG.0
03B8 0F;            0209         @IOCBRG
03B9 C203C5;        0210         LBZ OPWT30                  ..BRANCH IF NO ERRORS
03BC D4B453;        0211         CALL UCALL
03BF 28;            0212         DC CDERR
03C0 002B;          0213         DC (IOCBW1)..ELSE PRINT OUT MESSAGE
03C2 C003A3;        0214         LBR OPWT12
03C5 D5;            0215 OPWT30 EXIT
03C6 ;              0216  ..
03C6 ;              0217  ..
03C6 ;              0218  ..
```

```
03C6 ;              0219  ..    IOCB INITIALIZE ROUTINE
03C6 ;              0220  ..
03C6 ;              0221  ..
03C6 46BC;          0222 IOCBIN @R6!->IOCBPT.1          ..POINT RF @ IOCB
03C8 46AC;          0223        @R6!->IOCBPT.0
03CA 8CFC09AE;      0224        IOCBPT.0+WRITEP->TMPRQ1.0 ..ADVANCE POINTER TO WRITE PARM.
03CE 9C7C00BE;      0225        IOCBPT.1+"0->TMPRQ1.1
03D2 F8005E;        0226        0->@TMPRQ1              ..INIT. WRITE PARAMETER
03D5 1E;            0227        INC TMPRQ1
03D6 5E;            0228        ->@TMPRQ1
03D7 1E;            0229        INC TMPRQ1              ..POINT T1 @ UNIT NO.
03D8 5E;            0230        ->@TMPRQ1               ..DEFAULT OF 0
03D9 1E;            0231        INC TMPRQ1
03DA F809AD;        0232        9->TMPRQ2.0
03DD F8205E;        0233 LOOPIW SPACE->@TMPRQ1          ..BLANK OUT FILENAME & EXTENSION
03E0 1E;            0234        INC TMPRQ1
03E1 2D;            0235        DEC TMPRQ2
03E2 8D;            0236        TMPRQ2.0
03E3 CA03DD;        0237        LBNZ LOOPIW
03E6 8EFC03AE;      0238        TMPRQ1.0+3->TMPRQ1.0    ..POINT T1 @ FILE DEF.
03EA 9E7C00BE;      0239        TMPRQ1.1+"0->TMPRQ1.1
03EE F8025E;        0240        2->@TMPRQ1              ..INIT TO ASCII FILE
03F1 8EFC07AE;      0241        TMPRQ1.0+7->TMPRQ1.0    ..POINT T1 @ DEV. MNUMONIC
03F5 9E7C00BE;      0242        TMPRQ1.1+"0->TMPRQ1.1
03F9 F8445E;        0243        'D'->@TMPRQ1            ..SET DEVICE TO DISK
03FC 1E;            0244        INC TMPRQ1
03FD F84B5E;        0245        'K'->@TMPRQ1            ..IOCB INITIALIZED
0400 F802BD;        0246        A.1(PACKET)->TMPRQ2.1   ..SETUP SRNAM PACKET
0403 F89FAD;        0247        A.0(PACKET)->TMPRQ2.0
0406 F8025D;        0248        A.1(LINEBF)->@TMPRQ2    ..SETUP INPUT PARAMETER
0409 1D;            0249        INC TMPRQ2
040A F84F5D;        0250        A.0(LINEBF)->@TMPRQ2
040D 1D;            0251        INC TMPRQ2
040E 1D;            0252        INC TMPRQ2
040F 8EFF155D;      0253        TMPRQ1.0-21->@TMPRQ2    ..SETUP OUTPUT POINTER
0413 2D;            0254        DEC TMPRQ2
0414 9E7F005D;      0255        TMPRQ1.1-"0->@TMPRQ2
0418 D5;            0256        EXIT                    ..RETURN FROM ROUTINE
0419 ;              0257  ..
0419 ;              0258  ..
0419 ;              0259  ..
0419 ;              0260  ..      THESE ROUTINES CHECK FOR DISK ERRORS AND TAKE THE
0419 ;              0261  ..      APPROPRIATE ACTION.
0419 ;              0262  ..
0419 ;              0263  ..
0419 ;              0264  ..
0419 F800BE;        0265 CKRERR A.1(IOCBR+1)->TMPRQ1.1  ..POINT T1 TO READ STATUS
041C F808AE;        0266        A.0(IOCBR+1)->TMPRQ1.0
041F 0EFBC9;        0267        @TMPRQ1.XOR.IEOF        ..TEST FOR END OF FILE
0422 C20439;        0268        LBZ CKR10               ..BRANCH IF EOF
0425 0E;            0269        @TMPRQ1                 ..TEST FOR ERROR
0426 C20442;        0270        LBZ CKR20               ..BRANCH NO ERROR
0429 F802BE;        0271        A.1(ERRFLG)->TMPRQ1.1
042C F8A4AE;        0272        A.0(ERRFLG)->TMPRQ1.0
042F F8015E;        0273        1->@TMPRQ1              ..SET ERROR FLAG
0432 D4B453;        0274        CALL UCALL
0435 28;            0275        DC CDERR
0436 0007;          0276        DC (IOCBR)
0438 D5;            0277        EXIT
0439 F802BE;        0278 CKR10  A.1(NOTEOF)->TMPRQ1.1   ..SET NOT EOF FLAG
043C F8A3AE;        0279        A.0(NOTEOF)->TMPRQ1.0
043F F8015E;        0280        1->@TMPRQ1
0442 D5;            0281 CKR20  EXIT
0443 ;              0282  ..
0443 ;              0283  ..
0443 ;              0284  ..
0443 ;              0285  ..      CHECK WRITE ERROR FOR FILE #1
0443 ;              0286  ..
0443 ;              0287  ..
0443 ;              0288  ..
0443 F800BE;        0289 CKW1ER A.1(IOCBW1+1)->TMPRQ1.1 ..POINT T1 @ STATUS
0446 F82CAE;        0290        A.0(IOCBW1+1)->TMPRQ1.0
0449 0E;            0291        @TMPRQ1
044A C2045C;        0292        LBZ CKW110              ..BRANCH IF NO ERROR
044D F802BE;        0293        A.1(ERRFLG)->TMPRQ1.1
0450 F8A4AE;        0294        A.0(ERRFLG)->TMPRQ1.0
0453 F8015E;        0295        1->@TMPRQ1              ..SET ERROR FLAG
0456 D4B453;        0296        CALL UCALL
0459 28;            0297        DC CDERR
045A 002B;          0298        DC (IOCBW1)
045C D5;            0299 CKW110 EXIT
045D ;              0300
```

```
045D ;              0301          END BEGIN
0000
$U0000

C R O S S   R E F E R E N C E   L I S T I N G

SYMBOL          ADDR    DEF     REFERENCES

BEGIN           0000    U       0301
CDENT           001E            0040  0146
CDERR           0028            0035  0178  0212  0275  0297
CKR10           0439    0278    0268
CKR20           0442    0281    0270
CKRERR          0419    0265    0126  0140
CKW110          045C    0299    0292
CKW1ER          0443    0289    0135  0144
CLOSE           0002            0039  0138  0142
CP10            0305    0118    0136
CP20            0332    0137    0122  0131
CREAD           0012            0032  0156  0190
DEVICE          001F            0055
ENDBUF          0007            0051
ERRFLG          02A4    0083    0271  0272  0293  0294
FILEDF          0018            0054
GETSEC          0006            0037  0124
IEOF            00C9            0085  0267
*INPBUF         004F    0073    0064  0065  0070  0071
INPMSG          02A5    0091    0112
IOCBIN          03C6    0222    0153  0187
IOCBPT          000C            0087  0222  0223  0224  0225
IOCBR           0007    0061    0125  0139  0154  0172  0173  0174  0179  0265
                                0266  0276
IOCBRG          000F            0086  0173  0174  0175  0207  0208  0209
IOCBW1          002B    0067    0134  0143  0188  0206  0207  0208  0213  0289
                                0290  0298
LINEBF          024F    0074    0157  0191  0248  0250
LOOPIW          03DD    0233    0237
NOTEOF          02A3    0081    0118  0119  0127  0128  0278  0279
OPEN            0000            0036  0171  0205
OPENPR          0000            0048
OPENR           0348    0153    0113  0169
OPENW           0387    0187    0117  0203
OPRT12          0364    0166    0180
OPRT15          036D    0170    0165
OPRT30          0386    0181    0176
OPWT12          03A3    0200    0214
OPWT15          03AC    0204    0199
OPWT30          03C5    0215    0210
PACKET          029F    0075    0161  0195  0246  0247
PUTSEC          0010            0038  0133
RETYPE          02D6    0098    0168  0202
SPACE           0020            0056  0233
SRNAM           0024            0034  0160  0194
SRNERR          B452            0042  0162  0163  0196  0197
START           02F3    0110    0023  0024
STARTB          0005            0050
STATUS          0001            0049
TMPRG1          000E            0088  0118  0119  0120  0121  0127  0128  0129
                                0130  0162  0163  0164  0196  0197  0198  0224
                                0225  0226  0227  0228  0229  0230  0231  0233
                                0234  0238  0238  0239  0239  0240  0241  0241
                                0242  0242  0243  0244  0245  0253  0255  0265
                                0266  0267  0269  0271  0272  0273  0278  0279
                                0280  0289  0290  0291  0293  0294  0295
TMPRG2          000D            0089  0232  0235  0236  0246  0247  0248  0249



                                0250  0251  0252  0253  0254  0255
TMPRG3          000B            0090
TYPE            0014            0033  0111  0115  0167  0201
UCALL           B453            0041  0110  0114  0123  0132  0137  0141  0145
                                0155  0159  0166  0170  0177  0189  0193  0200
                                0204  0211  0274  0296
UNITNO          000B            0053
WRITEP          0009            0052  0224
WRTMG1          02C0    0095    0116
```

Appendix F
I/O Group Assignments

The I/O group number is transmitted by the OUT1 instruction. The transmitted group number remains in force until the next OUT1. Interim I/O instructions OUT2 through OUT7 and INP 2 through INP7 will be recognized only by those devices assigned to the current group number.

External flags EF1, EF2, EF3, and EF4 are conditioned by the group number, and change their meanings as that number changes.

GROUP 1 - $(0000\ 0001)_2$ - Terminal, Disk Printer

OUT2	Load data-terminal UART transmitter-holding register
OUT3	Load data-terminal UART control register
OUT6	RESERVED - Printer data out (parallel interface)
INP2	Read data-terminal UART receiver-holding register
INP3	Read data-terminal status register
EF1	Reserved for Printer
EF4	Data-terminal serial data in

GROUP 2 - $(0000\ 0010)_2$ - Reserved for MOPS

OUT2	Load MOPS UART transmitter-holding register
OUT3	Load MOPS UART control register
INP2	Read MOPS UART receiver-holding register
INP3	Read MOPS UART status register

GROUP 4 - $(0000\ 0100)_2$ - Reserved for PROM Program

OUT2	Low-order address bits to PROM
OUT3	High-order address bits to PROM
OUT4	Data to PROM
OUT5	Control to PROM
OUT6	Control to PROM
INP4	Read data from PROM
EF1	Switch S1 or PROM Programmer

Group 8 - (0000 1000) - Disk Controller

OUT4	Control byte to disk controller
OUT5	Control byte to disk controller
OUT7	DMA count to disk controller
INP4	Disk-controller status byte
INP5	Disk-controller results register
EF3	Disk-controller interrupt-identifier flag

Appendix G
Utility Program (UT71) Listing

```
!M
0000 ;                0001  ..*************************************************************************
0000 ;                0002
0000 ;                0003 ..NAME:   UT71, VER 0.0
0000 ;                0004 ..DATE:   8/2/83          CHANGED HLT FROM 76 TO 60 MSEC, FIXED WRITA  10/24/8
0000 ;                0005 ..
0000 ;                0006 ..        UT71 IS A MONITOR PROGRAM DESIGNED TO EXAMINE OR ALTER MEMORY,
0000 ;                0007 ..        TO BEGIN PROGRAM EXECUTION AT A GIVEN LOCATION, TO PROVIDE DISK,
0000 ;                0008 ..        TERMINAL, AND PRINTER INTERFACE ROUTINES, AND TO PROVIDE MEANS FOR
0000 ;                0009 ..        COMMUNICATING FROM TERMINAL TO DISK AND SYSTEM I/O.   THE MONITOR
0000 ;                0010 ..        COMMANDS ARE INITIATED BY TYPING D, F, I, M, S, P, T, L, B, W, R,
0000 ;                0011 ..        !, OR ?.  THE  FUNCTIONS INCLUDE MEMORY DISPLAY <D>, FILL <F>,
0000 ;                0012 ..        INSERT <I>, MOVE <M>, AND SUBSTITUTE <S>, RUN PROGRAM <P>, SELF
0000 ;                0013 ..        TEST <T>, LOAD OPERATING SYSTEM FROM DRIVE 0 <L> OR ANY DRIVE <B>,
0000 ;                0014 ..        WRITE <W> OR READ <R> A SECTOR, AND WRITE <!> OR READ <?> AN I/O
0000 ;                0015 ..        PORT.  ALSO INCLUDED ARE THE STANDARD READ AND TYPE ROUTINES FOR
0000 ;                0016 ..        COMMUNICATION WITH THE USERS TERMINAL, AND A PARALLEL PRINTER
0000 ;                0017 ..        OUTPUT ROUTINE.
0000 ;                0018 ..        UPON STARTING UT71, THE CONTENTS OF ALL CPU REGISTERS EXCEPT R0
0000 ;                0019 ..        AND R1 ARE SAVED IN RAM AT #8C00.
0000 ;                0020 ..        UT71 RESIDES IN ROM AT 8000-87FF, AND MUST HAVE RAM AT 8800-8FFF
0000 ;                0021 ..        (8C00-8CFF IF DISK ROUTINES NOT USED.)
0000 ;                0022 ..        UT71 PROVIDES MEANS FOR FORCING KEYBOARD READS TO GO TO A COMMAND
0000 ;                0023 ..        FILE INTERPETER INSTEAD.
0000 ;                0024 ..*************************************************************************
0000 ;                0025 ..         SYSTEM EQUATES, CONSTANTS & OFFSETS
0000 ;                0026 ..*************************************************************************
0000 ;                0027 ..
0000 ;                0028 ..        COMMAND FILE INTERPETER START ADDRESS (SEE READ ROUTINE)
0000 ;                0029 ..
0000 ;                0030 CFREAD  EQU  8FFDH
0000 ;                0031 ..
0000 ;                0032 ..        REGISTER ASSIGNMENTS - GEN. & UTILITIES
0000 ;                0033
0000 ;                0034 SP EQU 002H          ..STACK POINTER
0000 ;                0035 PC EQU 003H          ..PROGRAM COUNTER
0000 ;                0036 CALL EQU 004H        ..CALL ROUTINE REGISTER
0000 ;                0037 RETN EQU 005H        ..RETURN ROUTINE REGISTER
0000 ;                0038 LINK EQU 006H        ..SUBROUTINE.DATA LINK
0000 ;                0039 TMPRG1 EQU 007H      ..TEMPORARY REGISTER
0000 ;                0040 TMPRG2 EQU 008H      ..TEMPORARY REGISTER
0000 ;                0041 TMPRG3 EQU 009H      ..TEMPORARY REGISTER
0000 ;                0042 ADRPTR EQU 008H      ..HOLDS ADDR DURING STORE FROM TP
0000 ;                0043 CNT EQU 00AH         ..BYTE COUNT
0000 ;                0044 SRC EQU 00BH         ..SOURCE REGISTER
0000 ;                0045 TPTR EQU 00BH        ..TABLE POINTER
0000 ;                0046 DELAY EQU 00CH       ..DELAY ROUTINE COUNTER
0000 ;                0047 PTER EQU 00CH        ..IOCB PTR
0000 ;                0048 PTR EQU 00CH         ..IOCB PTR
0000 ;                0049 ASL EQU 00DH         ..HEX INPUT REGISTER
0000 ;                0050 DEST EQU 00DH        ..DESTINATION REGISTER
0000 ;                0051 AUX EQU 00EH         ..AUX.1 HOLDS BIT TIME CONSTANT
0000 ;                0052 CHAR EQU 00FH        ..STORES ASCII I/O
0000 ;                0053
0000 ;                0054 ..        RAM / ROM ALLOCATIONS
0000 ;                0055
0000 ;                0056 UT71 EQU 08000H
0000 ;                0057 TOPSTK EQU 08CFFH
0000 ;                0058 WRAM EQU 08C1FH
0000 ;                0059
0000 ;                0060          ASCII CONTROL CHARACTERS
```

```
0000 ;              0061
0000 ;              0062 NULL EQU 000H          ..NULL
0000 ;              0063 COMMA EQU 2CH          ..COMMA
0000 ;              0064 SEMCOL EQU 3BH         ..SEMICOLON
0000 ;              0065 BS EQU 008H            ..BACK SPACE
0000 ;              0066 LF EQU 00AH            ..LINE FEED
0000 ;              0067 CR EQU 00DH            ..CARRIAGE RETURN
0000 ;              0068 EOF EQU 013H           ..END OF FILE
0000 ;              0069 SPACE EQU 020H         ..SPACE
0000 ;              0070 CRLF EQU 00D0AH        ..CR LF
0000 ;              0071
0000 ;              0072 ..       CONSTANTS
0000 ;              0073 BDSEL EQU 001H         ..PORT FOR TWO LEVEL I/O SELECT
0000 ;              0074 LNECNT EQU 00FH        ..# OF BYTES PER LINE IN DISPLAY ROUTINE
0000 ;              0075 LINES EQU 014H         ..NUMBER OF LINES PER SCREEN LOAD
0000 ;              0076 PGMSRT EQU 00005H      ..START ADDRESS FOR INIT1
0000 ;              0077 PROMPT EQU 02AH        ..PROMPT CHARACTER
0000 ;              0078 ROWLEN EQU 028H        ..# OF CHARACTERS IN A ROW
0000 ;              0079 TRMINL EQU 001H        ..SELECTS THE TERMINAL
0000 ;              0080 UARTBD EQU 001H        ..SELECTS THE UART
0000 ;              0081 URTCTL EQU 003H        ..WRITE TO UART CONTROL REGISTER
0000 ;              0082 CTLWRD EQU 01DH        ..UART CONTROL WORD
0000 ;              0083 CHARAC EQU 002H        ..PORT FOR UART WORD OUT
0000 ;              0084 STATUS EQU 003H        ..PORT TO READ UART STATUS
0000 ;              0085
0000 ;              0086 ..*****************************************************************************
0000 ;              0087                  ORG UT71
8000 ;              0088
8000 71;            0089          DIS;
8001 00;            0090          DC 0                    ..DISABLE
8002 F880B0;        0091          LDI A.1(UT71) ;PHI R0..ESTABLISH PROGRAM COUNTER AT
8005 ;              0092                                  ..8000 HEX
8005 ;              0093
8005 ;              0094 ..*****************************************************************************
8005 ;              0095 ..                 REGISTER SAVE
8005 ;              0096 ..        SAVES CONTENTS OF THE CPU REGISTERS @#8C00.
8005 ;              0097 ..       CLOBBERS R0 AND R1 (LEAVES 0'S AS A REMINDER)
8005 ;              0098 ..*****************************************************************************
8005 ;              0099
8005 F88CB1;        0100          A.1(WRAM)->R1.1         ..TOP OF SAVE AREA
8008 F81FA1;        0101          A.0(WRAM)->R1.0
800B E1;            0102          SEX R1
800C 21;            0103 LOOP     DEC R1                  ..POINT BELOW WHERE SAVED R IS TO GO
800D F8D073;        0104          0D0H->@-                ..LOAD SEP R0 INSTRUCTION FOR RETURN
8010 81F6CF;        0105          R1.0/2;LSDF             ..FOR EVEN VALUES OF R1
8013 F910;          0106          ORI 10H                 ..MAKE 9X INSTRUCTION
8015 FC81;          0107          ADI 81H                 ..OTHERWISE 8X INSTRUCTION
8017 51;            0108          ->@R1                   ..STORE FOR EXECUTION
8018 F33A26;        0109          XOR;BNZ UT71A           ..LEAVE IF NO RAM THERE
801B D1;            0110          SEP R1                  ..GO EXECUTE
801C 73;            0111          ->@-                    ..AND STORE RESULT
801D 81FF033A0C;    0112          R1.0-3;BNZ LOOP         ..LOOP FOR REGISTERS F - 2
8022 73737351;      0113          ->@-,@-,@-,@R1          ..FILL LOCATIONS FOR 0 AND 1 WITH 0
8026 C08381;        0114 UT71A    LBR INIT
8029 ;              0115 ..
8029 ;              0116          ORG UT71+002CH          ..PRESERVE START ADDRESS
802C ;              0117 .
802C D480FE;        0118 START    CALL TIMALC
802F 9EFA01;        0119          AUX.1;ANI 01H           ..ECHO SET ?
8032 3241;          0120          BZ SCAN1                ..BRANCH IF YES
8034 ;              0121
8034 ;              0122 ..*****************************************************************************
8034 ;              0123 ..                    OUTPUT THE UTILITY PROMPT
8034 ;              0124 ..*****************************************************************************
8034 ;              0125
8034 ;              0126
8034 D483E8;        0127 PRMPT    CALL TPOFF              ..SEL GROUP 1
8037 D483F0;        0128          CALL OSTRNG             ..TYPE SCAN MODE '*' PROMPT
803A 0D0A;          0129          DC (CRLF)
803C 2A;            0130          DC PROMPT
803D 00;            0131          DC 0
803E ;              0132
803E ;              0133 ..*****************************************************************************
803E ;              0134 ..                MONITOR COMMAND INTERPRETER
803E ;              0135 ..       FETCHES THE ADDRESS FROM THE COMMAND TABLE AND SETS
803E ;              0136 ..       THE PROGRAM COUNTER TO IT
803E ;              0137 ..    REG USED:  PTR, CHAR, SP, ASL
```

```
803E ;              0138 ..*************************************************************************
803E ;              0139
803E ;              0140
803E D4;            0141 SCNLTR  SEP CALL;
803F 813E;          0142         DC (READ)                  ..READ COMMAND (LEAVES CHAR.  IN D)
8041 ;              0143
8041 9F52;          0144 SCAN1   CHAR.1->@SP                ..GET INPUT, STORE FOR COMPARE
8043 F85BAB;        0145         A.0(TAB2-2)->TPTR.0        ..INITIALIZE TABLE POINTER
8046 93BB;          0146         PC.1->TPTR.1
8048 1B1B;          0147 SCAN    INC TPTR;INC TPTR          ..PT TO NEXT (FIRST) ENTRY
804A 4B3285;        0148         LDA TPTR;BZ ERROR          ..ERROR IF END OF TABLE
804D F3;            0149         XOR                        ..LOOK FOR MATCH
804E 3A48;          0150         BNZ SCAN                   ..LOOP IF NOT
8050 BDAD;          0151         ->ASL.1;->ASL.0            ..ZERO CHARACTER REGISTER
8052 D481A220;      0152         CALL TYPE6;DC ' '          ..SPACE STARTS COMMAND
8056 2222;          0153         DEC SP;DEC SP              ..FAKE IT FOR THE RETURN
8058 ;              0154                                    ..PICK UP COMMAND ADDRESS
8058 4BB6;          0155         LDA TPTR;PHI LINK          ..AND TRANSFER TO THE
805A ;              0156                                    ..SUBROUTINE BY EXECUTING
805A 4BA6;          0157         LDA TPTR;PLO LINK          ..A RETURN INSTRUCTION
805C D5;            0158         SEP R5                     ..,P=3,X=2,R4 ;SEP CALL;,,R5 ;RETURN,R2=#8CFF
805D ;              0159
805D ;              0160 ..*************************************************************************
805D ;              0161 ..                    COMMAND TABLES
805D ;              0162 ..*************************************************************************
805D ;              0163
805D 44;            0164 TAB2    DC 'D'
805E 82BD;          0165         DC (DISPLY)                ..MEMORY DISPLAY
8060 49;            0166         DC 'I'
8061 83A7;          0167         DC (INSERT)                ..INSERT INTO MEMORY
8063 4D;            0168         DC 'M'
8064 82F7;          0169         DC (MOVE)                  ..MOVE A BLOCK OF MEMORY
8066 46;            0170         DC 'F'
8067 8240;          0171         DC (FILL)                  ..FILL A BLOCK OF MEMORY
8069 53;            0172         DC 'S'
806A 8099;          0173         DC (SUBST)                 ..BYTE SUBSTITUTION
806C 50;            0174         DC 'P'
806D 829F;          0175         DC (RUN)                   ..RUN A USER PROGRAM
806F 54;            0176         DC 'T'
8070 874E;          0177         DC (TEST)                  ..RUN MONITOR SELF TEST
8072 4C;            0178         DC 'L'
8073 8405;          0179         DC (LOAD)                  ..LOAD OPERATING SYSTEM FROM DRIVE 0
8075 42;            0180         DC 'B'
8076 8400;          0181         DC (BOOT)                  ..LOAD SAME FROM ANY DRIVE
8078 57;            0182         DC 'W'
8079 867A;          0183         DC (WDISK)                 ..UTILITY DISK WRITE
807B 52;            0184         DC 'R'
807C 867C;          0185         DC (RDISK)                 ..UTILITY DISK READ
807E 21;            0186         DC '!'
807F 86E2;          0187         DC (OUTPORT)               ..UTILITY OUTPUT TO PORT
8081 3F;            0188         DC '?'
8082 8707;          0189         DC (INPORT)                ..UTILITY INPUT FROM PORT
8084 00;            0190         DC 0
8085 ;              0191 ..
8085 ;              0192 ..*************************************************************************
8085 ;              0193 ..                    UTILITY ERROR MESSAGE
8085 ;              0194 ..            NOTE: ENTRY HERE RESETS STACK TO TOP
8085 ;              0195 ..REG USED:   CHAR
8085 ;              0196 ..*************************************************************************
8085 ;              0197
8085 ;              0198                ORG UT71+0085H
8085 ;              0199
8085 F8FFA2;        0200 ERROR   LDI A.0(TOPSTK);PLO SP
8088 F88CB2;        0201         LDI A.1(TOPSTK);PHI SP
808B D4;            0202         SEP CALL;
808C 83F0;          0203         DC (OSTRNG)
808E 0D0A;          0204         DC (CRLF)
8090 4552524F52;    0205         DC 'ERROR'
8095 00;            0206         DC 0
8096 C082AD;        0207 PRMPT1  LBR RENTER
8099 ;              0208
8099 ;              0209 ..*************************************************************************
8099 ;              0210 ..                    START OF SUBROUTINES
8099 ;              0211 ..*************************************************************************
8099 ;              0212
8099 ;              0213 ..              MONITOR SUBSTITUTE FUNCTION
8099 ;              0214 ..       DISPLAYS THE FIRST BYE FROM THE ADDRESS GIVEN FOLLOWED
```

```
8099 ;              0215 ..       BY A HYPHEN.  IF A HEX PAIR IS ENTERED FOLLOWED BY A SPACE,
8099 ;              0216 ..       IT IS SUBSTITUTED FOR THE BYTE DISPLAYED,  IF A SPACE IS
8099 ;              0217 ..       ENTERED THERE IS NO CHANGE.  IN EITHER CASE THE DATA BYTE FROM
8099 ;              0218 ..       THE NEXT ADDRESS WILL THEN BE DISPLAYED.  THE ROUTINE IS ENDED
8099 ;              0219 ..       BY ENTERING A RETURN.
8099 ;              0220 ..REG USED:  ASL, SRC, CHAR
8099 ;              0221 ..*****************************************************************************
8099 ;              0222
8099 D4;            0223 SUBST    SEP CALL;
809A 82F0;          0224          DC (READHX)              ..INPUT ADDRESS
809C 9DBB;          0225          GHI ASL;PHI SRC          ..SAVE START ADDRESS
809E 8DAB;          0226          GLO ASL;PLO SRC
80A0 ;              0227
80A0 9FFB0A;        0228 DECODE   GHI CHAR;XRI LF          ..FIRST NON-HEX MUST BE
80A3 32AF;          0229          BZ ADDOUT                ..A LINEFEED OR
80A5 FB07;          0230          XRI 007H                 ..TERMINATION OR
80A7 3296;          0231          BZ PRMPT1
80A9 FB2D;          0232          XRI 02DH                 ..A SPACE
80AB 32B7;          0233          BZ OLDDTA
80AD 3085;          0234          BR ERROR                 ..ELSE ERROR
80AF ;              0235
80AF D4;            0236 ADDOUT   SEP CALL;
80B0 83F0;          0237          DC (OSTRNG);
80B2 0D;            0238          DC CR
80B3 00;            0239          DC 0
80B4 D4;            0240          SEP CALL;
80B5 80CD;          0241          DC (OUT1)
80B7 ;              0242
80B7 DC;            0243 OLDDTA   SEP DELAY;
80B8 17;            0244          DC 017H                  ..WAIT TO FINISH READ
80B9 0BBF;          0245          LDN SRC;PHI CHAR         ..STAY ON SAME LINE
80BB D4;            0246          SEP CALL;
80BC 81AE;          0247          DC (TYPE2)               ..HEX OUTPUT
80BE D4;            0248          SEP CALL;
80BF 83F0;          0249          DC (OSTRNG)              ..OUTPUT A HYPHEN
80C1 2D;            0250          DC '-'
80C2 00;            0251          DC 0
80C3 0BAD;          0252          LDN SRC;PLO ASL          ..COPY DATA FROM CELL INTO ASL
80C5 ;              0253
80C5 D4;            0254 GETDTA   SEP CALL;
80C6 82F0;          0255          DC (READHX)              ..GET ANY CHANGE
80C8 8D5B;          0256          GLO ASL;STR SRC          ..RESTORE THE DATA INTO THE CELL
80CA 1B;            0257          INC SRC                  ..OPEN THE NEXT CELL
80CB 30A0;          0258          BR DECODE                ..EXAMINE INPUT
80CD ;              0259
80CD DC;            0260 OUT1     SEP DELAY;
80CE 17;            0261          DC 017H
80CF 9BBF;          0262          GHI SRC;PHI CHAR         ..ROUTINE TO OUTPUT A HEX PAIR
80D1 D4;            0263          SEP CALL;
80D2 81AE;          0264          DC (TYPE2)               ..AND A SPACE
80D4 8BBF;          0265          GLO SRC;PHI CHAR
80D6 D4;            0266          SEP CALL;
80D7 81AE;          0267          DC (TYPE2)
80D9 D4;            0268          SEP CALL;
80DA 83F0;          0269          DC (OSTRNG);
80DC 20;            0270          DC SPACE
80DD 00;            0271          DC 0
80DE D5;            0272          SEP R5
80DF ;              0273
80DF ;              0274 ..*****************************************************************************
80DF ;              0275 ..       USER CALLABLE ROUTINE TO GENERATE A DELAY.  THE DELAY
80DF ;              0276 ..       CONSTANT IS PASSED AS AN INLINE PARAMETER.  THE CALL
80DF ;              0277 ..       IS MADE BY DOING A SEP RC
80DF ;              0278 ..REG USED:   DELAY, PC
80DF ;              0279 ..*****************************************************************************
80DF ;              0280
80DF ;              0281                          ORG UT71+00EEH
80EE ;              0282
80EE D3;            0283 DEXIT    SEP PC
80EF ;              0284
80EF 43;            0285 DELAY1   LDA PC
80F0 ;              0286
80F0 FF01;          0287 DELAY2   SMI 1
80F2 32EE;          0288          BZ DEXIT
80F4 30F0;          0289          BR DELAY2
80F6 ;              0290
```

```
80F6 ;              0291 ..*************************************************************************
80F6 ;              0292 ..      SETS UP POINTER TO THE DELAY ROUTINE, SETS UP THE UART,
80F6 ;              0293 ..      AND SETS UP ECHO (AUX.1=0) OR NO ECHO (AUX.1=1).
80F6 ;              0294 ..*************************************************************************
80F6 ;              0295
80F6 ;              0296
80F6 ;              0297                          ORG UT71+00FEH
80FE ;              0298
80FE 93BC;          0299 TIMALC  GHI PC;PHI DELAY        ..DELAY SUBROUTINE ADDRESS
8100 F8EF;          0300         LDI A.0(DELAY1)
8102 AC;            0301         PLO DELAY
8103 E3;            0302         SEX PC
8104 61;            0303         OUT BDSEL;
8105 01;            0304         DC UARTBD               ..SELECT GROUP 1
8106 63;            0305         OUT URTCTL;
8107 1D;            0306         DC CTLWRD               ..SET UP UART
8108 E2;            0307         SEX SP
8109 ;              0308
8109 F800BE;        0309 ECHOTST 0->AUX.1                ..DEFAULT TO ECHO (FULL DUPLEX)
810C D483F0;        0310         CALL OSTRNG
810F 0D0A55543731;  0311         DC CR,LF,'UT71 VER 0.0' ..PRINT UT71 VERSION NUMBER
8115 205645522030;
811B 2E30;
811D 0D0A;          0312         DC CR,LF
811F 2A;            0313         DC PROMPT               ..TYPE INITIAL '*' PROMPT
8120 00;            0314         DC 0
8121 D4813E;        0315         CALL READ               ..WAIT FOR RESPONSE
8124 9FFB0A;        0316         CHAR.1;XRI LF           ..CHECK FOR LF
8127 322A;          0317         BZ NOECHO               ..BRANCH IF YES
8129 D5;            0318         EXIT
812A F801BE;        0319 NOECHO  1->AUX.1                ..SET NO ECHO (HALF DUPLEX)
812D D5;            0320         EXIT
812E ;              0321
812E ;              0322
812E ;              0323 ..*************************************************************************
812E ;              0324 ..       READS ONE BYTE INTO CHAR.1.   WHEN ENTERED
812E ;              0325 ..       VIA READAH, ANY HEX INPUT IS ASSEMBLED
812E ;              0326 ..       INTO ASL AND DF =1, ELSE DF = 0 ON RETURN.
812E ;              0327 ..REG USED:  CHAR & ASL (AUX.1 HOLDS ECHO AND READ SOURCE FLAGS)
812E ;              0328 ..*************************************************************************
812E ;              0329
812E ;              0330                          ORG UT71+012FH
812F ;              0331
812F FC07;          0332 CKDEC   ADI 7                   ..CHECK FOR ASCII DECIMAL
8131 3337;          0333         BDF NFND                ..OUT OF RANGE
8133 FC0A;          0334         ADI 00AH                ..SUBTRACT NET 30
8135 3369;          0335         BDF FND
8137 ;              0336
8137 FC00;          0337 NFND    ADI 0                   ..SET DF = 0
8139 ;              0338
8139 9F;            0339 REXIT   GHI CHAR                ..PUT INPUT INTO D
813A D5;            0340         SEP R5                  .. & RETURN
813B ;              0341
813B F80038;        0342 READAH  LDI 0;SKP               ..SKIP TO READ1 WITH D=0
813E ;              0343
813E 93;            0344 READ    GHI PC                  ..CONSTANT > 0
813F AF;            0345         PLO CHAR                ..SAVE ENTRY POINT
8140 ;              0346
8140 9EFE;          0347         AUX.1*2                 ..IF COMMAND FILE IS IN CONTROL,
8142 C38FFD;        0348         LBDF CFREAD             ..GO TO IT, SKIP KEYBOARD READ
8145 ;              0349
8145 6BF6;          0350 READ1   INP STATUS;/2
8147 3B45;          0351         BNF READ1
8149 6A;            0352         INP CHARAC              ..READ CHARACTER
814A FA7FBF;        0353         ANI 07FH;->CHAR.1
814D 3245;          0354         BZ READ1                ..IGNORE IF ITS'S A NULL
814F 9EF6;          0355 CFRET   AUX.1;/2                ..COMMAND FILE RETURN POINT
8151 335B;          0356         BDF NEXT
8153 6BFE;          0357 RDWAIT  INP STATUS;*2           ..ECHO IF ECHO BIT SET
8155 3B53;          0358         BNF RDWAIT
8157 9F52;          0359         CHAR.1->@SP
8159 62;            0360         OUT CHARAC
815A 22;            0361         DEC SP
815B ;              0362
815B ;              0363
815B 8F;            0364 NEXT    GLO CHAR                ..CHECK ENTRY
815C 3A39;          0365         BNZ REXIT               ..ENTERED VIA READ
```

```
815E ;              0366
815E 9FFF41;        0367 CKHXE   GHI CHAR;SMI 041H        ..CHECK FOR ASCII HEX
8161 3B2F;          0368         BNF CKDEC                ..CHECK FOR ASCII DECIMAL
8163 FF06;          0369         SMI 6                    ..A THRU F
8165 3337;          0370         BDF NFND                 ..NO
8167 FC10;          0371         ADI 010H                 ..SUBTRACT NET 37
8169 ;              0372
8169 FA0F73;        0373 FND     .AND.0FH->@-             ..SAVE TEMPORARILY
816C 9D;            0374         GHI ASL                  ..SHIFT DATA INTO ASL
816D FEFEFEFE52;    0375         SHL;SHL;SHL;SHL;STR SP   ..SHL 4X
8172 8D;            0376         GLO ASL
8173 F6F6F6F6;      0377         SHR;SHR;SHR;SHR
8177 F1BD;          0378         OR;PHI ASL
8179 8D;            0379         GLO ASL
817A FEFEFEFE;      0380         SHL;SHL;SHL;SHL
817E 12;            0381         INC SP
817F F1AD;          0382         OR;PLO ASL
8181 FF00;          0383         SMI 0                    ..SET DF = 1
8183 3039;          0384         BR REXIT
8185 ;              0385
8185 ;              0386 ..*****************************************************************
8185 ;              0387 ..       TYPES ONE BYTE FROM CHAR.1 AS AN ASCII
8185 ;              0388 ..       CHARACTER OR AS TWO HEX DIGITS.  LINE FEEDS
8185 ;              0389 ..       ARE FOLLOWED BY SIX NULLS.  USES REGISTER
8185 ;              0390 ..       CHAR AND A STACK LOCATION.
8185 ;              0391 ..       @SP-1 HOLDS OUTPUT CHARACTER.
8185 ;              0392 ..       CHAR.0 HOLDS THE NUMBER OF BITS (11) IN
8185 ;              0393 ..       ITS LOWER DIGIT AND THE FOLLOWING CODE IN
8185 ;              0394 ..       ITS UPPER DIGIT:
8185 ;              0395 ..             0 -  BYTE OUTPUT
8185 ;              0396 ..             1 -  FIRST HEX OUTPUT
8185 ;              0397 ..             2 -  LAST NULL OUTPUT
8185 ;              0398 ..             8 -  LF OUTPUT
8185 ;              0399 ..
8185 ;              0400 ..REG USED:  CHAR (AUX.1 HOLDS ECHO AND READ SOURCE FLAGS)
8185 ;              0401 ..*****************************************************************
8185 ;              0402
8185 ;              0403                     ORG UT71+0198H
8198 ;              0404
8198 30A4;          0405 TYPED   BR TYPE
819A ;              0406                     ORG UT71+019CH
819C ;              0407
819C 30A0;          0408 TYPE5D  BR TYPE5
819E ;              0409
819E ;              0410                     ORG UT71+019FH
819F ;              0411
819F D5;            0412 TEXIT   SEP R5
81A0 ;              0413
81A0 45;            0414 TYPE5   LDA R5                   ..PICK UP DATA
81A1 38;            0415         SKP
81A2 ;              0416
81A2 46;            0417 TYPE6   LDA R6                   ..PICK UP DATA
81A3 38;            0418         SKP
81A4 ;              0419
81A4 9F73;          0420 TYPE    GHI CHAR;->@-            ..KEEP A COPY
81A6 FB0A;          0421         XRI LF                   ..IS IT A LINE-FEED ?
81A8 3AC0;          0422         BNZ TY2
81AA F880;          0423         LDI 080H                 ..# BITS ADI  # NULLS
81AC 30C2;          0424         BR TY3
81AE 9FF6F6F6F6;    0425 TYPE2   GHI CHAR;SHR;SHR;SHR;SHR
81B3 FCF6;          0426         ADI 0F6H                 ..CONVERT TO HEX
81B5 3BB9;          0427         BNF TY1                  ..IF A OR >, ADD 37
81B7 FC07;          0428         ADI 7
81B9 ;              0429
81B9 FFC673;        0430 TY1     SMI 0C6H;->@-            ..ELSE ADD 30
81BC F810;          0431         LDI 010H                 ..10 ADI  NO. OF BITS
81BE 30C2;          0432         BR TY3
81C0 ;              0433
81C0 F800;          0434 TY2     LDI 0                    ..NO OF BITS
81C2 ;              0435
81C2 AF;            0436 TY3     PLO CHAR
81C3 ;              0437
81C3 6BFE;          0438 BEGIN   INP STATUS;#2
81C5 3BC3;          0439         BNF BEGIN
81C7 12;            0440         INC SP                   ..PT BACK TO CHARACTER
81C8 62;            0441         OUT CHARAC
81C9 22;            0442         DEC SP
```

```
81CA ;              0443
81CA 8FFCF0AF;      0444 NXCHAR  GLO CHAR;ADI OFOH;PLO CHAR
81CE 3B9F;          0445         BNF TEXIT                  ..SEP R5 IF NO MORE
81D0 FF10;          0446         SMI 010H                   ..TEST FOR ALTERNATIVES
81D2 329F;          0447         BZ TEXIT                   ..TYPED LAST NULL
81D4 3BDA;          0448         BNF HEX1                   ..TYPED FIRST HEX
81D6 F800;          0449         LDI 0                      ..TYPED LF OR NULL
81D8 30E5;          0450         BR HEX3
81DA ;              0451
81DA 9FFA0F;        0452 HEX1    GHI CHAR;ANI 00FH          ..GET 2ND HEX DIGIT
81DD FCF6;          0453         ADI 0F6H                   ..CONVERT TO HEX
81DF 3BE3;          0454         BNF HEX2                   ..IF A OR MORE,
81E1 FC07;          0455         ADI 7                      ..ADD NET 37
81E3 ;              0456
81E3 FFC6;          0457 HEX2    SMI 0C6H                   ..ELSE ADD NET 30
81E5 ;              0458
81E5 73;            0459 HEX3    ->@-                       ..AND SAVE
81E6 30C3;          0460         BR BEGIN
81E8 ;              0461
81E8 ;              0462
81E8 ;              0463 ..*****************************************************************************
81E8 ;              0464 ..                    INPUT OPTION
81E8 ;              0465 ..          ALLOWS ENTRY OF EITHER STARTING AND ENDING
81E8 ;              0466 ..          ADDRESSES OR BYTE COUNT.  SEP R5S WITH THE
81E8 ;              0467 ..          STARTING ADDRESS IN REG SRC AND THE BYTE
81E8 ;              0468 ..          COUNT IN REG CNT.  RETURNS WITH  DF =1
81E8 ;              0469 ..          IF SYNTAX ERROR EXISTS.
81E8 ;              0470 ..REG USED:  ASL, SRC, CHAR, CNT
81E8 ;              0471 ..*****************************************************************************
81E8 ;              0472
81E8 ;              0473                  ORG UT71+0200H
8200 ;              0474
8200 D4;            0475 OPTION  SEP CALL;
8201 82F0;          0476         DC (READHX)                ..GET THE STARTING ADDRESS
8203 9DBB;          0477         GHI ASL;PHI SRC            ..AND SAVE IT
8205 8DAB;          0478         GLO ASL;PLO SRC
8207 F800ADBD;      0479         LDI 0;PLO ASL;PHI ASL      ..CLEAR THE INPUT REG.
820B 9FFB20;        0480         GHI CHAR;XRI SPACE         ..FIRST NONSMI HEX MUST
820E 3231;          0481         BZ CNTIN                   ..BE A SPACE OR
8210 FB0D;          0482         XRI 00DH                   ..A HYPHEN
8212 3A46;          0483         BNZ PRMPT2                 ..ELSE SYNTAX ERROR
8214 D4;            0484         SEP CALL;
8215 82F0;          0485         DC (READHX)                ..EXPECT ENDING ADDRESS
8217 ;              0486
8217 8B52;          0487 BYTCNT  GLO  SRC;STR SP            ..CALCULATE THE BYTE COUNT
8219 8DF7AA;        0488         GLO ASL;SM;PLO CNT
821C 9B52;          0489         GHI SRC;STR SP
821E 9D77BA;        0490         GHI ASL;SMB;PHI CNT
8221 333F;          0491         BDF EXITOK                 ..CHECK FOR SRC < ASL
8223 ;              0492
8223 8D52;          0493 INVERT  GLO ASL;STR SP             ..ELSE EXCHANGE THE CONTENTS OF
8225 8BAD;          0494         GLO SRC;PLO ASL            ..SRC AND ASL
8227 02AB;          0495         LDN SP;PLO SRC
8229 9D52;          0496         GHI ASL;STR SP
822B 9BBD;          0497         GHI SRC;PHI ASL
822D 02BB;          0498         LDN SP;PHI SRC
822F 3017;          0499         BR  BYTCNT                 ..RECALCULATE
8231 ;              0500
8231 ;              0501
8231 D4;            0502 CNTIN   SEP CALL;
8232 82F0;          0503         DC (READHX)                ..INPUT THE BYTE COUNT
8234 8DFF01AA;      0504         GLO ASL;SMI 1;PLO CNT
8238 9D7F00BA;      0505         GHI ASL;SMBI 0;PHI CNT
823C 333F;          0506         BDF EXITOK
823E 1A;            0507         INC CNT
823F D5;            0508 EXITOK  SEP R5                     ..RETURN WHEN DONE
8240 ;              0509
8240 ;              0510 ..*****************************************************************************
8240 ;              0511 ..                        FILL ROUTINE
8240 ;              0512 ..          LOADS MEMORY BEGINNING AT ADDRESS CONTAINED
8240 ;              0513 ..          IN SRC WITH DATA CONTAINED IN ASL.0 FOR
8240 ;              0514 ..          THE NUMBER OF BYTES SPECIFIED BY CNT.
8240 ;              0515 ..          USER CALLABLE @ USRFIL.
8240 ;              0516 ..REG USED:  ASL, SRC, CNT ,CHAR
8240 ;              0517 ..*****************************************************************************
8240 ;              0518
8240 D4;            0519 FILL    SEP CALL;
```

```
8241 8303;          0520          DC (READAD)              ..GET THE ADDRESSES
8243 D4;            0521          SEP CALL;
8244 824B;          0522          DC (USRFIL)              ..CALL THE MOVE
8246 ;              0523
8246 C082AD;        0524 PRMPT2   LBR RENTER               ..GOTO UT71 AND PROMPT
8249 ;              0525
8249 1B;            0526 NXTCEL   INC SRC                  ..POINT TO NEXT CELL
824A 2A;            0527          DEC CNT                  ..REDUCE BYTE COUNT
824B 8D5B;          0528 USRFIL   GLO ASL;STR SRC          ..LOAD THE DATA;USER ENTRY PT.
824D 8A;            0529          GLO CNT                  ..LOOP UNTIL COUNT = 0
824E 3A49;          0530          BNZ NXTCEL
8250 9A;            0531          GHI CNT
8251 3A49;          0532          BNZ NXTCEL
8253 D5;            0533          SEP R5                   ..EXIT THE CALL
8254 ;              0534
8254 ;              0535 ..*****************************************************************************
8254 ;              0536 ..               MOVE ROUTINE
8254 ;              0537 ..       COPIES A BLOCK OF MEMORY FORM ONE CONTINUOUS AREA
8254 ;              0538 ..       TO ANOTHER CONTINUOUS AREA IN MEMORY.  THERE IS NO
8254 ;              0539 ..       RESTRICTION AS TO THE DIRECTION OF THE MOVE AND THE
8254 ;              0540 ..       AREAS MAY OVERLAP.
8254 ;              0541 ..       REG USED:  SRC, DEST, CHAR, & CNT
8254 ;              0542 ..*****************************************************************************
8254 ;              0543
8254 E2;            0544 USRMOV   SEX SP
8255 8B52;          0545          GLO SRC;STR SP           ..TEST THE RELATIVE POSITION
8257 8DF7;          0546          GLO DEST;SM              ..OF SOURCE & DESTINATION
8259 3A61;          0547          BNZ DIRECT               ..NOT EQUAL!
825B 9B52;          0548          GHI SRC;STR SP           ..RETURN IF THEY ARE EQUAL
825D 9D77;          0549          GHI DEST;SMB
825F 329D;          0550          BZ USRBYE                ..EXIT TO CALLER
8261 ;              0551
8261 8B52;          0552 DIRECT   GLO SRC;STR SP           ..ELSE TEST FOR UP OR DOWN
8263 8DF7;          0553          GLO DEST;SM              ..DIRECTION OF THE MOVE
8265 9B52;          0554          GHI SRC;STR SP
8267 9D77;          0555          GHI DEST;SMB
8269 3378;          0556          BDF MOVUP
826B 0B5D;          0557 MOVDN    LDN SRC;STR DEST         ..DO THE MOVE DOWN AND
826D 8A;            0558          GLO CNT                  ..AND CHECK IF DONE
826E 3A73;          0559          BNZ MOVDN1
8270 9A;            0560          GHI CNT
8271 329D;          0561          BZ USRBYE                ..EXIT TO CALLER
8273 ;              0562
8273 1B1D;          0563 MOVDN1   INC SRC;INC DEST         ..ADJUST THE POINTERS
8275 2A;            0564          DEC CNT                  ..REDUCE THE BYTE COUNT
8276 306B;          0565          BR MOVDN                 ..FINISHED
8278 ;              0566
8278 8A52;          0567 MOVUP    GLO  CNT;STR SP          ..SET THE POINTERS TO THE
827A 8BF4AB;        0568          GLO SRC;ADD;PLO SRC      ..TOP OF MOVE AREAS
827D 9A52;          0569          GHI CNT;STR SP
827F 9B74BB;        0570          GHI SRC;ADC;PHI SRC
8282 8A52;          0571          GLO CNT;STR SP
8284 8DF4AD;        0572          GLO DEST;ADD;PLO DEST
8287 9A52;          0573          GHI CNT;STR SP
8289 9D74BD;        0574          GHI DEST;ADC;PHI DEST
828C 3B90;          0575          BNF UP
828E ;              0576
828E 309E;          0577 ERRGO    BR USRBYE+1              ..EXIT DF=1 IF OVERFLOW
8290 ;              0578
8290 0B5D;          0579 UP       LDN SRC;STR DEST         ..DO THE MOVE UP
8292 8A;            0580          GLO CNT                  ..AND CHECK IF DONE
8293 3A98;          0581          BNZ UP1
8295 9A;            0582          GHI CNT
8296 329D;          0583          BZ USRBYE                ..EXIT TO CALLER
8298 ;              0584
8298 2B2D2A;        0585 UP1      DEC SRC;DEC DEST;DEC CNT..ADJUST THE POINTERS
829B 3090;          0586          BR UP
829D ;              0587
829D F6;            0588 USRBYE   SHR                      ..SET DF=0 IF A<FFFF
829E D5;            0589          SEP R5                   ..EXIT TO CALLER
829F ;              0590
829F ;              0591 ..*****************************************************************************
829F ;              0592 ..       STARTS A USER PROGRAM WITH SPECIFIED ADDRESS
829F ;              0593 ..       IN REGISTER 0 AND X=0.
829F ;              0594 ..REG USED:  CHAR, ASL, R0
829F ;              0595 ..*****************************************************************************
829F ;              0596
```

```
829F D4;          0597 RUN      SEP CALL;
82A0 82F0;        0598          DC (READHX)              ..LOOK FOR STARTING ADDRESS
82A2 FB0D;        0599          XRI CR                   ..FIRST NON-HEX MUST BE A
82A4 CA8085;      0600          LBNZ ERROR               ..CR, ELSE SYNTAX ERROR
82A7 ;            0601
82A7 9DB0;        0602 RUN1     GHI ASL;PHI RO           ..GET THE ADDRESS
82A9 BDA0;        0603          GLO ASL;PLO RO
82AB E0;          0604          SEX RO
82AC D0;          0605          SEP RO                   ..AND GO!
82AD ;            0606
82AD ;            0607 ..*****************************************************************************
82AD ;            0608 ..              GENERAL REENTER ROUTINE
82AD ;            0609 ..*****************************************************************************
82AD ;            0610
82AD F8B4A0;      0611 RENTER   LDI A.0(RENTR1);PLO RO   ..CAN BE ENTERED WITH X AND P
82B0 F882B0;      0612          LDI A.1(RENTR1);PHI RO   ..SET TO ANYTHING AND RESETS
82B3 D0;          0613          SEP RO                   ..ALL THE SCRT REGISTERS
82B4 F834A3;      0614 RENTR1   LDI A.0(PRMPT);PLO PC
82B7 F880B3;      0615          LDI A.1(PRMPT);PHI PC
82BA C0838F;      0616          LBR ENTER2
82BD ;            0617
82BD ;            0618 ..*****************************************************************************
82BD ;            0619 ..                      OUTPUT
82BD ;            0620 ..        FORMATS AND OUTPUTS MEMORY DATA BEGINNING
82BD ;            0621 ..        AT THE  ADDRESS IN REG SRC FOR THE NUMBER
82BD ;            0622 ..        OF BYTES SPECIFIED IN REG CNT
82BD ;            0623 ..REG USED:  SRC, CNT, CHAR
82BD ;            0624 ..*****************************************************************************
82BD ;            0625
82BD D4;          0626 DISPLY   SEP CALL;
82BE 8200;        0627          DC (OPTION)              ..GET STARTING ADDRESS
82C0 FB0D;        0628          XRI CR                   ..TERMINATE WITH CR
82C2 CA8085;      0629          LBNZ ERROR
82C5 ;            0630
82C5 D4;          0631 OUTPUT   SEP CALL;
82C6 83F0;        0632          DC (OSTRNG);
82C8 0A;          0633          DC LF
82C9 00;          0634          DC 0                     ..START ON A NEW LINE
82CA D4;          0635          SEP CALL;
82CB 80CD;        0636          DC (OUT1)                ..OUTPUT THE ADDRESS OF THE
82CD ;            0637                                   ..CURRENTLY OPENED CELL
82CD ;            0638
82CD D4;          0639 SPCOUT   SEP CALL;
82CE 83F0;        0640          DC (OSTRNG)
82D0 20;          0641          DC SPACE
82D1 00;          0642          DC 0
82D2 ;            0643
82D2 4BBF;        0644 DATOUT   LDA SRC;PHI CHAR         ..RETRIEVE THE CELL DATA
82D4 D4;          0645          SEP CALL;
82D5 81AE;        0646          DC (TYPE2)               ..AND OUTPUT IT
82D7 8A;          0647          GLO CNT                  ..DETERMINE IF THE
82D8 3ADD;        0648          BNZ NOTDON               ..REQUESTED NO. OF BYTES
82DA 9A;          0649          GHI CNT                  ..HAVE BEEN SENT
82DB 3246;        0650          BZ PRMPT2                ..GET A NEW COMMAND
82DD ;            0651
82DD 2A;          0652 NOTDON   DEC CNT                  ..DEC THE BYTE COUNT
82DE 8BFA0F;      0653          GLO SRC;ANI LNECNT
82E1 3AEB;        0654          BNZ SAMELN               ..END OF CURRENT LINE?
82E3 D4;          0655          SEP CALL;
82E4 83F0;        0656          DC (OSTRNG)
82E6 3B;          0657          DC ';'
82E7 0D;          0658          DC CR
82E8 00;          0659          DC 0
82E9 30C5;        0660          BR OUTPUT
82EB ;            0661
82EB F6;          0662 SAMELN   SHR
82EC 33D2;        0663          BDF DATOUT               ..WITHIN PAIR
82EE 30CD;        0664          BR  SPCOUT               ..ELSE BETWEEN PAIRS
82F0 ;            0665
82F0 ;            0666 ..*****************************************************************************
82F0 ;            0667 ..       FILLS ASL AS LONG AS HEX DIGITS ARE ENTERED
82F0 ;            0668 ..*****************************************************************************
82F0 ;            0669
82F0 D4;          0670 READHX   SEP CALL;
82F1 813B;        0671          DC (READAH)
82F3 33F0;        0672          BDF READHX
82F5 D5;          0673          SEP R5
```

```
82F6 ;              0674
82F6 ;              0675 ..*****************************************************************************
82F6 ;              0676 ..       MOVE COMMAND
82F6 ;              0677 ..       CALLS USRMOV AND REQUESTS SRC&DEST ADDR'S
82F6 ;              0678 ..*****************************************************************************
82F6 ;              0679
82F6 ;              0680                 ORG UT71+02F7H
82F7 ;              0681
82F7 D4;            0682 MOVE     SEP CALL;
82F8 8303;          0683          DC (READAD)            ..GET SRC&DEST ADDR'S
82FA D4;            0684          SEP CALL;
82FB 8254;          0685          DC (USRMOV)            ..DO THE MOVE
82FD C38085;        0686          LBDF ERROR             ..ERROR IF OVER FFFF ON MOVE
8300 C08246;        0687          LBR PRMPT2             ..IF OK,GOTO UT71 PROMPT
8303 ;              0688
8303 ;              0689 ..*****************************************************************************
8303 ;              0690 ..     SUBROUTINE TO GET THE ADDRESSES FOR OTHER ROUTINES
8303 ;              0691 ..*****************************************************************************
8303 ;              0692
8303 D4;            0693 READAD   SEP CALL;
8304 8200;          0694          DC (OPTION)            ..DETERMINE THE MODE
8306 FB20;          0695          XRI SPACE              ..MUST BE A SPACE
8308 3A60;          0696          BNZ ERR1               ..ELSE ERROR
830A ADBD;          0697          PLO ASL;PHI ASL        ..CLEAR INPUT REGISTER
830C D4;            0698          SEP CALL;
830D 82F0;          0699          DC (READHX)            ..INPUT THE CONSTANT
830F FB0D;          0700          XRI CR                 ..'CR' TERMINATES
8311 3A60;          0701          BNZ ERR1               ..ELSE ERROR
8313 D5;            0702          SEP R5
8314 ;              0703
8314 ;              0704 ..*****************************************************************************
8314 ;              0705 ..      FILLS ASL UNTIL A CARRIAGE RETURN IS ENTERED
8314 ;              0706 ..*****************************************************************************
8314 ;              0707
8314 D4;            0708 READCR   SEP CALL;
8315 813B;          0709          DC (READAH)
8317 FB0D;          0710          XRI CR
8319 3A14;          0711          BNZ READCR
831B D5;            0712          SEP R5
831C ;              0713
831C ;              0714 ..*****************************************************************************
831C ;              0715 ..               OSTRNG
831C ;              0716 ..*****************************************************************************
831C ;              0717
831C F8EFAC;        0718 MSGE     LDI 0EFH;PLO DELAY
831F F880BC;        0719          LDI 080H;PHI DELAY
8322 ;              0720
8322 46BF;          0721 MSGE1    LDA LINK;PHI CHAR
8324 322B;          0722          BZ EXITM
8326 D4;            0723          SEP CALL;
8327 8198;          0724          DC (TYPED)
8329 3022;          0725          BR MSGE1
832B ;              0726
832B D5;            0727 EXITM    SEP R5
832C ;              0728
832C ;              0729 ..*****************************************************************************
832C ;              0730 ..       THIS ROUTINE PRINTS TO THE LINE PRINTER THE CONTENTS OF RF.1.
832C ;              0731 ..       IT SUPRESSES PRINTING OF THE LINE FEEDS, AND REPLACES CARRIAGE
832C ;              0732 ..       RETURNS WITH A CR-LF PAIR. NORMALLY, THIS ROUTINE RETURNS WITH
832C ;              0733 ..       THE DFLAG SET. BUT IF THE CHARACTER IN RF.1 WAS A DC3 (END OF
832C ;              0734 ..       FILE), THE DFLAG WILL BE RESET ON RETURN.
832C ;              0735 ..*****************************************************************************
832C ;              0736
832C 9FFB0A;        0737 PRNTRF   GHI RF;XRI 0AH         ..IF LINE FEED, EXIT
832F 3248;          0738          BZ EXITDF
8331 9FFB13;        0739          GHI RF;XRI 13H         ..IF DC3, EXIT
8334 324C;          0740          BZ EXITEF
8336 9FFBFF52;      0741 PRINT1   GHI RF;XRI 0FFH;STR R2 ..INVERT DATA
833A 343A;          0742          B1 $                   ..WAIT UNTIL READY
833C 6622;          0743          OUT 6;DEC R2           ..OUTPUT CHARACTER
833E 9FFB0D;        0744          GHI RF;XRI 0DH         ..CARRIAGE RETURN ?
8341 3A48;          0745          BNZ EXITDF             ..NO, EXIT
8343 FB0ABF;        0746          LDI 0AH;PHI RF         ..YES, PRINT A LINE FEED
8346 3036;          0747          BR PRINT1
8348 FB01F6;        0748 EXITDF   LDI 1;SHR              ..SET DFLAG
834B D5;            0749          EXIT
834C F6;            0750 EXITEF   SHR                    ..RESET DFLAG
```

```
834D D5;          0751          EXIT
834E ;            0752
834E ;            0753 ..*************************************************************************
834E ;            0754                ORG UT71+035DH
835D ;            0755
835D C082AD;      0756 PRMPT5  LBR RENTER
8360 C08085;      0757 ERR1    LBR ERROR              ..GENERAL FOR THIS PAGE
8363 ;            0758
8363 ;            0759 ..*************************************************************************
8363 ;            0760 ..DESC:   STANDARD SEP CALL;,A( AND RETURN
8363 ;            0761 ..REG USED:   SP,PC,SEP CALL;,A(,RETURN,LINK & STACK
8363 ;            0762 ..*************************************************************************
8363 ;            0763
8363 ;            0764                ORG UT71+0363H
8363 ;            0765
8363 ;            0766                    STANDARD CALL
8363 ;            0767
8363 D3;          0768 EXITC   SEP PC                 ..GO TO IT
8364 ;            0769
8364 E2;          0770 CALLR   SEX SP                 ..SET R(X)
8365 9673;        0771         GHI LINK;STXD          ..SAVE THE CURRENT LINK ON
8367 8673;        0772         GLO LINK;STXD          ..THE STACK
8369 93B6;        0773         GHI PC;PHI LINK
836B 83A6;        0774         GLO PC;PLO LINK
836D 46B3;        0775         LDA LINK;PHI PC        ..PICK UP THE SUBROUTINE
836F 46A3;        0776         LDA LINK;PLO PC        ..ADDRESS
8371 3063;        0777         BR EXITC
8373 ;            0778
8373 ;            0779                  ..STANDARD RETURN
8373 ;            0780
8373 D3;          0781 EXITR   SEP PC                 ..RETURN TO MAIN PGM
8374 ;            0782
8374 96B3;        0783 RETR    GHI LINK;PHI PC
8376 86A3;        0784         GLO LINK;PLO PC
8378 E212;        0785         SEX SP;INC SP          ..SET THE STACK POINTER
837A 72A6;        0786         LDXA;PLO LINK          ..RESTORE THE CONTENTS OF
837C F0B6;        0787         LDX;PHI LINK           ..LINK
837E 9F;          0788         GHI CHAR               ..PUT THE CONTENTS OF CHAR.1 INTO D
837F ;            0789                                ..BEFORE RETURNING
837F 3073;        0790         BR EXITR
8381 ;            0791
8381 ;            0792 ..*************************************************************************
8381 ;            0793 ..              REGISTER INITIALIZATION ROUTINE
8381 ;            0794 ..       INITIALIZES REGISTER C TO THE DELAY ROUTINE, REG 2 AS A STACK
8381 ;            0795 ..       POINTER TO LOCATION 8CFF HEX, REG 4 TO CALL, REG 5 TO RETURN
8381 ;            0796 ..       AND REG 3 AS PROGRAM COUNTER. FOR ENTER1 REG 3 IS 0005, FOR
8381 ;            0797 ..       ENTER2 REG 3 MUST BE PRESET.
8381 ;            0798 ..       REG USED:  PC, DELAY, CALL, RETURN, SP
8381 ;            0799 ..*************************************************************************
8381 ;            0800
8381 F82CA3;      0801 INIT    LDI A.0(START);PLO PC
8384 F880B3;      0802         LDI A.1(START);PHI PC
8387 308F;        0803         BR ENTER2
8389 ;            0804
8389 F805A3;      0805 ENTER1  LDI A.0(PGMSRT);PLO PC
838C F800B3;      0806         LDI A.1(PGMSRT);PHI PC
838F ;            0807
838F F8EFAC;      0808 ENTER2  LDI A.0(DELAY1);PLO DELAY   ..DELAY ROUTINE
8392 F880BC;      0809         LDI A.1(DELAY1);PHI DELAY
8395 F883B4B5;    0810         LDI A.1(CALLR);PHI CALL;PHI RETN
8399 F864A4;      0811         LDI A.0(CALLR);PLO CALL
839C F874A5;      0812         LDI A.0(RETR);PLO RETN
839F F8FFA2;      0813         LDI A.0(TOPSTK);PLO SP
83A2 F88CB2;      0814         LDI A.1(TOPSTK);PHI SP
83A5 E2D3;        0815         SEX SP;SEP PC
83A7 ;            0816
83A7 ;            0817 ..*************************************************************************
83A7 ;            0818 ..              HEX BYTE INSERT ROUTINE
83A7 ;            0819 ..       INSERTS HEX PAIRS INTO MEMORY STARTING AT A SPECIFIED
83A7 ;            0820 ..       ADDRESS. AFTER A ";" ALL IS IGNORED UNTIL A RETURN
83A7 ;            0821 ..       THEN A NEW ADDRESS IS EXPECTED. ANY NON-HEX DATA IS
83A7 ;            0822 ..       IGNORED BETWEEN HEX PAIRS BUT NOTHING IS PERMITTED
83A7 ;            0823 ..       BETWEEN MEMBERS OF THE PAIR. ROUTINE IS TERMINATED
83A7 ;            0824 ..       WITH A RETURN, EXCEPT AFTER A ";".
83A7 ;            0825 ..       REG USED:  ASL, SRC, CHAR
83A7 ;            0826 ..*************************************************************************
83A7 ;            0827
```

```
83A7 F800BDAD;      0828 INSERT  0->ASL.1,ASL.0          ..CLEAR INPUT REGISTER
83AB D4813B;        0829 INSERT1 CALL READAH
83AE 3BAB;          0830         BNF INSERT1             ..IGNORE INPUTS UNTIL FIRST HEX
83B0 D482F0;        0831         CALL READHX;            ..THEN INPUT UNTIL FIRST NON-HEX
83B3 FB203A60;      0832         .XOR.' ';BNZ ERR1       ..IT MUST BE A SPACE
83B7 9DBB;          0833         ASL.1->SRC.1
83B9 8DAB;          0834         ASL.0->SRC.0            ..INPUTS WERE THE STARTING ADDRESS
83BB ;              0835
83BB D4813B;        0836 NXTCHR  CALL READAH
83BE 3BCA;          0837         BNF NTDATA              ..IF NEXT INPUT IS HEX
83C0 D4813B;        0838         CALL READAH             ..GET A SECOND
83C3 3B60;          0839         BNF ERR1                ..WHICH MUST ALSO BE HEX
83C5 8D5B1B;        0840         ASL.0->@SRC;INC SRC     ..AND STORE HEX PAIR INTO MEMORY
83C8 30BB;          0841         BR NXTCHR               ..LOOK FOR MORE
83CA ;              0842
83CA FB0D325D;      0843 NTDATA  .XOR.CR;BZ PRMPT5       ..IF INPUT WAS CR, LEAVE
83CE FB36;          0844         .XOR.('; '.XOR.CR)
83D0 3ADB;          0845         BNZ COMCHK              ..IF INPUT WAS ';',
83D2 D48314;        0846         CALL READCR             ..IGNORE EVERYTHING UNTIL CR
83D5 D481A20A;      0847         CALL TYPE6;DC LF        ..ADD LINEFEED
83D9 30A7;          0848         BR INSERT               ..START NEXT LINE WITH NEW ADDRESS
83DB ;              0849
83DB FB17;          0850 COMCHK  .XOR.(',' .XOR.';')
83DD 3ABB;          0851         BNZ NXTCHR              ..IF INPUT WAS ',',
83DF D48314;        0852         CALL READCR             ..IGNORE EVERYTHING UNTIL CR
83E2 D481A20A;      0853         CALL TYPE6;DC LF        ..ADD LINEFEED
83E6 30BB;          0854         BR NXTCHR               ..START NEW LINE WITHOUT NEW ADDRESS
83E8 ;              0855 ..
83E8 ;              0856 ..*****************************************************************************
83E8 ;              0857 ..                    SELECT GROUP 1
83E8 ;              0858 ..*****************************************************************************
83E8 ;              0859
83E8 E3;            0860 TPOFF   SEX PC
83E9 61;            0861         OUT BDSEL
83EA 01;            0862         DC TRMINL
83EB D5;            0863         SEP R5
83EC ;              0864
83EC ;              0865 ..*****************************************************************************
83EC ;              0866 ..                UTILITY ENTRY TABLE
83EC ;              0867 ..*****************************************************************************
83EC ;              0868
83EC ;              0869             ORG UT71+03F0H
83F0 ;              0870
83F0 C0831C;        0871 OSTRNG  LBR MSGE
83F3 C08389;        0872 INIT1   LBR ENTER1
83F6 C0838F;        0873 INIT2   LBR ENTER2
83F9 C082AD;        0874 GOUT71  LBR RENTER
83FC C0815E;        0875 CKHEX   LBR CKHXE
83FF ;              0876
83FF ;              0877
83FF ;              0878 ..*****************************************************************************
83FF ;              0879 ..DISK I/O ROUTINES
83FF ;              0880 ..*****************************************************************************
83FF ;              0881
83FF ;              0882 ..REGISTER EQUATES
83FF ;              0883
83FF ;              0884 DMAPTR      EQU  0                 ..DMA POINTER
83FF ;              0885 INTPC       EQU  1                 ..INTERRUPT PC
83FF ;              0886 IOCBPTR     EQU  7                 ..IOCB POINTER
83FF ;              0887 CMDCNT      EQU  8                 ..NO. OF COMMAND WORDS
83FF ;              0888 TRKCNT      EQU  9                 ..TRACK COUNT DURING LOAD
83FF ;              0889 PARA        EQU  0FH               ..PARAMETER BLOCK POINTER
83FF ;              0890
83FF ;              0891 ..RAM EQUATES
83FF ;              0892
83FF ;              0893 RAMADR      EQU  UT71+800H         ..BEGINNING OF RAM
83FF ;              0894 IOCB        EQU  8F00H             ..LOADER IOCB
83FF ;              0895 STA0        EQU  8F10H             ..STATUS 0
83FF ;              0896
83FF ;              0897 ..MICRODOS EQUATES
83FF ;              0898
83FF ;              0899 STK         EQU  0BFFFH            ..TOP OF STACK
83FF ;              0900 CAL         EQU  0918CH            ..CALL ADDRESS
83FF ;              0901 RET         EQU  0919CH            ..RETURN ADDRESS
83FF ;              0902 ECHOTP      EQU  0910AH            ..ECHO STATUS
83FF ;              0903 ENTRY       EQU  09040H            ..ENTRY ADDRESS INTO MICRODOS
83FF ;              0904 MICRO       EQU  0A843H            ..'MICRODOS' NAME IN OP SYS
```

```
83FF ;              0905
83FF ;              0906 ..DISK BOARD I/O EQUATES
83FF ;              0907
83FF ;              0908 DISKSEL     EQU  1                    ..BOARD SELECTION
83FF ;              0909 DMASEL      EQU  4                    ..DMA DIRECTION SELECT
83FF ;              0910 TERMCNT     EQU  4                    ..TERMINAL COUNT/ABORT
83FF ;              0911 NECSTA      EQU  4                    ..MAIN STATUS REGISTER
83FF ;              0912 COMMAND     EQU  5                    ..COMMAND/DATA REGISTER
83FF ;              0913 DATA        EQU  5                    ..COMMAND/DATA REGISTER
83FF ;              0914 BYTECNT     EQU  7                    ..BYTE COUNT SELECT
83FF ;              0915
83FF ;              0916 ..NEC COMMANDS
83FF ;              0917
83FF ,              0918 SPCMD       EQU 03H                   ..SPECIFY
83FF ;              0919 RCCMD       EQU 07H                   ..RECALIBRATE
83FF ;              0920 SKCMD       EQU 0FH                   ..SEEK
83FF ;              0921 RDCMD       EQU 46H                   ..READ
83FF ;              0922 WTCMD       EQU 45H                   ..WRITE
83FF ;              0923 SISCMD      EQU 08H                   ..SENSE INTERRUPT STATUS
83FF ;              0924 INVCMD      EQU 00H                   ..INVALID COMMAND
83FF ;              0925
83FF ;              0926 ..DISK DATA CONSTANTS
83FF ;              0927
83FF ;              0928 BC          EQU  04H                  ..BYTE COUNT
83FF ;              0929 N           EQU  02H                  ..N
83FF ;              0930 EOT         EQU  09H                  ..EOT
83FF ;              0931 GPL3        EQU  1BH                  ..GPL3
83FF ;              0932 DTL         EQU  0FFH                 ..DTL
83FF ;              0933 FM          EQU  40H                  ..DENSITY
83FF ;              0934 SRT         EQU  10H                  ..STEP RATE: 15 MS
83FF ;              0935 HLT         EQU  3CH                  ..HEAD LOAD TIME: 60 MS
83FF ;              0936 HUT         EQU  0FH                  ..HEAD UNLOAD TIME: 240 MS
83FF ;              0937 DMA         EQU  00H                  ..DMA OPERATION
83FF ;              0938 DMANOP      EQU  00H                  ..NO DMA OPERATION
83FF ;              0939 CRCREAD     EQU  01H                  ..CRC READ
83FF ;              0940 DMAO        EQU  02H                  ..DISK WRITE
83FF ;              0941 DMAI        EQU  03H                  ..DISK READ
83FF ;              0942 RCA         EQU  01H                  ..GROUP SELECT 1
83FF ;              0943 NEC         EQU  08H                  ..GROUP SELECT 8
83FF ;              0944 MAXTRK   EQU     70                   ..NUMBER OF TRACKS ON SONY
83FF ;              0945 MAXSEC   EQU     09                   ..9 SECTORS / TRACK
83FF ;              0946
83FF ;              0947 ..*****************************************************************************
83FF ;              0948 ..
83FF ;              0949 ..                    LOAD MICRODOS
83FF ;              0950 ..
83FF ;              0951                    ORG UT71+0400H
8400 ;              0952 ..
8400 D4813B3B00;    0953 BOOT     CALL READAH;BNF BOOT          ..GET DRIVE # (WAIT UNTIL HEX KEY)
8405 82AF92BF;      0954 LOAD     SP.0->PARA.0;SP.1->PARA.1     ..ENTER FROM "L", ASSUME DRIVE = 0
8409 8D73;          0955          ASL.0->@-                     ..DRIVE # IS @ PARA
840B FAFCCA8085;    0956          .AND.0FCH;LBNZ ERROR          ..ERROR IF DRIVE > 3
8410 A0F890B0;      0957          ->DMAPTR.0;90H->DMAPTR.1      ..BEGINNING OF MICRODOS AREA
8414 F824A9;        0958          A.0(TKTABL)->TRKCNT.0
8417 F884B9;        0959          A.1(TKTABL)->TRKCNT.1         ..TABLE FOR MICRODOS LOAD
841A D48499;        0960          CALL SPECIFY                  ..SET UP DRIVE PARAMETERS
841D 302E;          0961          BR LOAD1
841F ;              0962 ..*****************************************************************************
841F ;              0963
841F ;              0964                          ORG UT71+421H
8421 ;              0965
8421 C08254;        0966          LBR USRMOV                    ..FOR COMPATIBILITY WITH UT62
8424 ;              0967                                        ..MOVE COMMAND
8424 ;              0968
8424 ;              0969 ..*****************************************************************************
8424 ;              0970
8424 01021C;        0971 TKTABL   DC 1,2,28                     ..TABLE CONTAINS TRACK #, STARTING
8427 020024;        0972          DC 2,0,36                     ..SECTOR # (-1), AND BYTE COUNT (X128)
842A 030020;        0973          DC 3,0,32                     ..FOR ALL MICRODOS LOAD
842D 00;            0974          DC 0                          ..0 ENDS TABLE
842E D4859C;        0975 LOAD1    CALL SEEKA                    ..FIRST SEEK IS RECAL (ASL.1 WAS 0)
8431 8D3A7E;        0976          ASL.0;BNZ NOLOAD              ..EXIT IF ABNORMAL TERM.
8434 49;            0977          @TRKCNT!
8435 BD;            0978 LOAD2    ->ASL.1                       ..SETUP TRACK #
8436 49AD;          0979          @TRKCNT!->ASL.0               ..AND SECTOR #
8438 D4859C;        0980          CALL SEEKA                    ..SEEK ALSO SETS UP FOR READ
843B 8D3A7E;        0981          ASL.0;BNZ NOLOAD              ..EXIT IF ABNORMAL TERM.
```

```
843E F801A7;        0982          A.0(IOCB+1)->IOCBPTR.0
8441 F88FB7;        0983          A.1(IOCB+1)->IOCBPTR.1          ..POINT TO BYTE COUNT IN IOCB
8444 4957;          0984          @TRKCNT!->@IOCBPTR              ..LOAD IT FROM TABLE
8446 D485FB;        0985          CALL READA                      ..GO READ
8449 8D3A7E;        0986          ASL.0;BNZ NOLOAD                ..EXIT IF ABNORMAL TERM.
844C 493A35;        0987          @TRKCNT!;BNZ LOAD2              ..REPEAT FOR ALL TRACKS
844F ;              0988
844F ;              0989 ..*********************************************************************
844F ;              0990
844F ;              0991
844F F843A7;        0992 MICTST   A.0(MICRO)->IOCBPTR.0           ..POINT AT 'MICRODOS' IN OP SYS
8452 F8A8B7;        0993          A.1(MICRO)->IOCBPTR.1
8455 E7;            0994          SEX IOCBPTR
8456 47;            0995          @IOCBPTR!                       ..GET 'M' AND POINT X AT 'I'
8457 F3;            0996          XOR                             ..EXCLUSIVE OR 'M' AND 'I'
8458 17;            0997          INC IOCBPTR                     ..POINT AT 'C'
8459 F3;            0998          XOR                             ..EXCLUSIVE OR 'M/I' AND 'C'
845A FB47;          0999          XRI 47H                         ..WAS 'MIC' THERE ?
845C 3A7E;          1000          BNZ NOLOAD                      ..OP SYS IS NOT IN MEMORY
845E ;              1001
845E ;              1002 ..*********************************************************************
845E ;              1003
845E ;              1004
845E D483F0;        1005 LOADOK   CALL OSTRNG
8461 0D0A0A00;      1006          DC ODOAH,OAH,OO
8465 F8FFA2;        1007          A.0(STK)->SP.0                  ..SET UP OP SYS STACK POINTER,
8468 F8BFB2;        1008          A.1(STK)->SP.1                  ..CALL, RETURN, AND ECHO STATUS
846B F88CA4;        1009          A.0(CAL)->CALL.0
846E F89CA5;        1010          A.0(RET)->RETN.0
8471 F80AA7;        1011          A.0(ECHOTP)->IOCBPTR.0
8474 F891B4;        1012          A.1(CAL)->CALL.1
8477 B5;            1013          ->RETN.1
8478 B7;            1014          ->IOCBPTR.1
8479 9E57;          1015          AUX.1->@IOCBPTR
847B C09040;        1016          LBR ENTRY                       ..GO TO OPERATING SYSTEM
847E ;              1017
847E ;              1018
847E ;              1019 ..*********************************************************************
847E ;              1020
847E D483F0;        1021 NOLOAD   CALL OSTRNG                     ..PRINT NOT LOADED MESSAGE
8481 0A4D494352;    1022          DC LF,'MICRODOS NOT LOADED',0
8486 4F444F53204E;
848C 4F54204C4F41;
8492 44454400;
8496 C082AD;        1023          LBR RENTER                      ..GO BACK TO MONITOR (FIXES SP)
8499 ;              1024
8499 ;              1025 ..*********************************************************************
8499 ;              1026 ..
8499 ;              1027 ..       SPECIFY SETS UP DRIVE PARAMETERS
8499 ;              1028 ..
8499 F804A7;        1029 SPECIFY  A.0(IOCB+4)->IOCBPTR.0          ..RESET POINTER
849C F88FB7;        1030          A.1(IOCB+4)->IOCBPTR.1
849F E7;            1031          SEX IOCBPTR
84A0 F83C73;        1032          LDI (HLT.OR.DMA);->@-           ..HEAD LOAD TIME AND DMA
84A3 F81F73;        1033          LDI (SRT.OR.HUT);->@-           ..STEP RATE AND HEAD UNLOAD TIME
84A6 F80373;        1034          SPCMD->@-                       ..SPECIFY COMMAND
84A9 F8FF73;        1035          OFFH->@-                        ..CLEAR BYTE COUNT
84AC F80073;        1036          DMANOP->@-                      ..DMANOP
84AF F803A8;        1037          3->CMDCNT.0                     ..BYTES IN COMMAND SEQUENCE
84B2 D484CD;        1038          CALL CMD
84B5 D5;            1039          EXIT
84B6 ;              1040 ..*********************************************************************
84B6 ;              1041 ..
84B6 ;              1042 ..       RECAL RECALIBRATES DRIVE:  ENTER WITH DRIVE #, IN DMAPTR.0
84B6 ;              1043 ..       WAIT MUST BE USED AFTER RECAL
84B6 ;              1044 ..
84B6 F803A7;        1045 RECAL    A.0(IOCB+3)->IOCBPTR.0          ..POINT AT UNIT IN IOCB
84B9 F88FB7;        1046          A.1(IOCB+3)->IOCBPTR.1
84BC E7;            1047          SEX IOCBPTR
84BD 8073;          1048          DMAPTR.0->@-                    ..HEAD AND UNIT
84BF F80773;        1049          RCCMD->@-                       ..RECALIBRATE COMMAND
84C2 27;            1050          DEC IOCBPTR
84C3 F80073;        1051          DMANOP->@-                      ..DMANOP
84C6 F802A8;        1052          2->CMDCNT.0
84C9 D484CD;        1053          CALL CMD
84CC D5;            1054          EXIT
84CD ;              1055 ..*********************************************************************
```

```
84CD ;              1056 ..      COMMAND ROUTINE OUTPUTS COMMAND WORDS FROM @ IOCB FOR NUMBER
84CD ;              1057 ..      SPECIFIED IN CMDCNT.0.  IT CLEARS SERVICE REQUEST FROM 765 FIRST,
84CD ;              1058 ..      AND CLEARS OUT ANY RESULTS THAT ARE PENDING.
84CD ;              1059 ..
84CD E3;            1060 CMD     SEX PC
84CE 6108;          1061         OUT DISKSEL;DC NEC              ..SELECT DISK GROUP
84D0 3EDA;          1062         BN3 CMD1                        ..IF 765 WANTS SERVICE
84D2 6CFE3BD2;      1063 CMD2    INP NECSTA;*2;BNF CMD2          ..WAIT FOR RQM
84D6 6508;          1064         OUT DATA;DC SISCMD              ..DO SENSE INTERRUPT COMMAND
84D8 C4C4;          1065 CMD4    NOP;NOP                         ..WAIT FOR 765
84DA 6CFE3BDA;      1066 CMD1    INP NECSTA;*2;BNF CMD1          ..WAIT FOR RQM
84DE FE3BE4;        1067         *2;BNF CMD3                     ..IF 765 WANTS TO OUTPUT STATUS,
84E1 6D30D8;        1068         INP DATA;BR CMD4                ..GET STATUS BYTES UNTIL CLEAR
84E4 36D2;          1069 CMD3    B3 CMD2                         ..LOOP IF SERVICE STILL WANTED
84E6 F800A7;        1070         A.0(IOCB)->IOCBPTR.0            ..POINT AT IOCB
84E9 F88FB7;        1071         A.1(IOCB)->IOCBPTR.1
84EC E7;            1072         SEX IOCBPTR
84ED 64;            1073         OUT DMASEL                      ..DIRECTION OF DMA
84EE 67;            1074         OUT BYTECNT                     ..LATCH BYTE COUNT
84EF E2;            1075 CMD6    SEX SP
84F0 6CFE3BF0;      1076 CMD5    INP NECSTA;*2;BNF CMD5          ..WAIT FOR RQM
84F4 E7;            1077         SEX IOCBPTR
84F5 65;            1078         OUT COMMAND                     ..OUTPUT COMMAND WORDS
84F6 28;            1079         DEC CMDCNT                      ..UNTIL COUNTER = 0
84F7 883AEF;        1080         CMDCNT.0; BNZ CMD6
84FA E3;            1081         SEX PC
84FB 6101;          1082         OUT DISKSEL;DC RCA              ..SELECT GROUP 1
84FD D5;            1083         EXIT
84FE ;              1084 ..*************************************************************************
84FE ;              1085 ..
84FE ;              1086 ..      FOR COMPATABILITY WITH UT21 LINE PRINTER ROUTINE
84FE ;              1087 ..
84FE ;              1088         ORG UT71+050EH
850E ;              1089 ..
850E C0832C;        1090         LBR PRNTRF
8511 ;              1091 ..*************************************************************************
8511 ;              1092 ..      WAIT ROUTINE WAITS FOR SERVICE REQUEST FROM 765, TIMES OUT IF NONE.
8511 ;              1093 ..      IF RESULT OF READ/WRITE, INPUTS STATUS BYTES.  IF RESULT OF SEEK OR
8511 ;              1094 ..      RECAL, DOES  SENSE INTERRUPT STATUS FIRST.  IF WRONG DRIVE, REPEATS.
8511 ;              1095 ..      USES CRC DMA CYCLE TO CLEAR DMA REQUEST IN CASE OF SERIOUS OVER-RUN.
8511 ;              1096
8511 F8FFB7;        1097 WAIT    OFFH->IOCBPTR.1                 ..SET UP TIMER
8514 E3;            1098 WAIT2   SEX PC
8515 6108;          1099         OUT DISKSEL;DC NEC
8517 361D;          1100         B3 WAIT1                        ..SERVICE DISK IF EF3
8519 2797;          1101         DEC IOCBPTR;IOCBPTR.1           ..IF NO REQUEST IN 1.3 SEC,
851B 3A14;          1102         BNZ WAIT2                       ..(2.5 SEC IF 2.5 MHZ CLK)
851D F810A7;        1103 WAIT1   A.0(STA0)->IOCBPTR.0            ..SET UP RESULT POINTER
8520 F88FB7;        1104         A.1(STA0)->IOCBPTR.1
8523 F88057;        1105         80H->@IOCBPTR                   ..EXIT WITH TERM ERROR
8526 3E6D;          1106         BN3 ENDWAIT
8528 ;              1107
8528 E7;            1108 WAIT5   SEX IOCBPTR                     ..START RESULT SERVICE
8529 6CFE3B29;      1109 WAIT3   INP NECSTA;*2;BNF WAIT3         ..WAIT FOR RQM
852D FA203A37;      1110         ANI 20H;BNZ WAIT4               ..IF BUSY BIT LOW (END OF SEEK)
8531 E3;            1111         SEX PC
8532 6508;          1112         OUT COMMAND;DC SISCMD           ..DO SENSE INTERRUPT COMMAND
8534 C43028;        1113         NOP;BR WAIT5                    ..AND SERVICE THAT RESULT
8537 ;              1114
8537 6D52;          1115 WAIT4   INP DATA;->@SP                  ..FIRST STATUS IS ST0 (SAVE)
8539 60C4;          1116 WAIT8   IRX;NOP
853B 6CFE3B3B;      1117 WAIT6   INP NECSTA;*2;BNF WAIT6         ..WAIT FOR RQM
853F FE3B45;        1118         *2;BNF WAIT7                    ..IF DIRECTION BIT STILL = IN,
8542 6D3039;        1119         INP DATA;BR WAIT8               ..INPUT STATUS AND LOOK AGAIN
8545 ;              1120
8545 F803A7;        1121 WAIT7   A.0(IOCB+3)->IOCBPTR.0
8548 F88FB7;        1122         A.1(IOCB+3)->IOCBPTR.1          ..WHEN ALL RESULTS INPUT, CHECK
854B 02F3FA03;      1123         @SP.XOR.@.AND.3                 ..IF DRIVE NUMBER FROM ST0 MATCHES
854F 3A11;          1124         BNZ WAIT                        ..THAT FROM COMMAND.  REPEAT IF NO
8551 E3;            1125         SEX PC
8552 02FAC0;        1126         @SP.AND.0C0H                    ..IF COMMAND TERMINATION (FROM ST0)
8555 326D;          1127         BZ ENDWAIT                      ..WASN'T NORMAL
8557 6500;          1128         OUT COMMAND;DC INVCMD           ..DO INVALID COMMAND
8559 C4C4;          1129         NOP;NOP                         ..(WANTS ONE RESULT)
855B 6CFE3B5B;      1130 WAIT9   INP NECSTA;*2;BNF WAIT9         ..WAIT FOR RQM
855F 6401;          1131         OUT DMASEL;DC CRCREAD           ..ENABLE PHONY DMA IN CIRCUIT
8561 6400;          1132         OUT DMASEL;DC DMANOP            ..(INPUTS RESULT IF DRQ HUNG HIGH
```

```
8563 C4C4;          1133          NOP;NOP                              ..FROM DMA OVER-RUN)
8565 6CFE3B65;      1134 WAIT10   INP NECSTA;*2;BNF WAIT10             ..WAIT FOR RQM
8569 FE3B6D;        1135          *2;BNF ENDWAIT                       ..IF DRQ DIDN'T INPUT RESULT
856C 6D;            1136          INP DATA                             ..DO IT ANYWAY
856D 6101;          1137 ENDWAIT  OUT DISKSEL;DC RCA                   ..BACK TO TERMINAL GROUP
856F D5;            1138          EXIT                                 ..EXIT DISK SERVICE ROUTINE
8570 ;              1139
8570 ;              1140 ..*****************************************************************************
8570 ;              1141
8570 ;              1142                    ..SEEK ROUTINE
8570 ;              1143
8570 ;              1144 ..WHEN ENTERED AT SEEKST, CALCULATES TRACK, SECTOR NO. FROM @ PARA +1 AND +2.
8570 ;              1145 ..CAN ALSO BE ENTERED AT SEEKA WITH ASL.1 = TRACK NO., ASL.0 = SECTOR NO.-1.
8570 ;              1146 ..(STILL USES @PARA FOR UNIT #)
8570 ;              1147 ..DOES RECAL IF TRACK = 0.  SETS UP IOCB FOR LATER READS OR WRITES
8570 ;              1148 ..
8570 ;              1149 ..       PARAMETER BLOCK POINTER:  @ PARA = UNIT NO.
8570 ;              1150 ..                                 @ +1   = PSN HIGH BYTE, @ +2 = LOW BYTE
8570 ;              1151 ..                                 @ +3   = BUFFER ADDRESS H.B., @ +4 = L.B.
8570 ;              1152 ..       SEEKST USES IOCBPTR.0 FOR COUNTER,.1 FOR TEMP STORE,
8570 ;              1153 ..       AUX.0 AS DIVIDEND H.B., ASL.1 FOR TRACK #, ASL.0 FOR SECTOR # (-1)
8570 ;              1154 ..       ASL.0 HOLDS TERMINATION RESULT AT END
8570 ;              1155 ..
8570 F800BD;        1156 SEEKST   0->ASL.1                             ..CLEAR FUTURE RESULT
8573 1F4FAE;        1157          INC PARA;@PARA!->AUX.0               ..PSN HIGH BYTE
8576 0FAD;          1158          @PARA->ASL.0                         ..AND LOW BYTE
8578 2F2F;          1159          DEC PARA;DEC PARA
857A F80273F84052;  1160          2->@-;40H->@SP                       ..DIVISOR = 9 SHIFTED LEFT 6 TIMES
8580 F807A7;        1161          7->IOCBPTR.0                          ..SUBTRACT AND SHIFT 7 TIMES
8583 8DF7B7;        1162 SUBLP    ASL.0-@->IOCBPTR.1
8586 128E77;        1163          INC SP;AUX.0-"@                      ..DIVIDEND - DIVISOR
8589 3B8E;          1164          BM SHRES                             ..IF NOT -
858B AE97AD;        1165          ->AUX.0;IOCBPTR.1->ASL.0              ..STORE NEW DIVIDEND
858E 9D7EBD;        1166 SHRES    ASL.1*2"->ASL.1                      ..SHIFT NO BORROW INTO RESULT
8591 F0F673F07652;  1167          @/2->@-;@/2"->@SP                    ..SHIFT DIVISOR RIGHT
8597 27;            1168          DEC IOCBPTR
8598 873A83;        1169          IOCBPTR.0;BNZ SUBLP                   ..LOOP 7 TIMES
859B 12;            1170          INC SP                               ..FIX STACK POINTER
859C ;              1171 ..
859C F80AA7;        1172 SEEKA    A.0(IOCB+10)->IOCBPTR.0       ..POINT AT IOCB DTL VALUE
859F F88FB7;        1173          A.1(IOCB+10)->IOCBPTR.1
85A2 E7;            1174          SEX IOCBPTR
85A3 F8FF73;        1175          DTL->@-                       ..DTL
85A6 F81B73;        1176          GPL3->@-                      ..GPL3
85A9 F80973;        1177          EOT->@-                       ..EOT
85AC F80273;        1178          N->@-                         ..N
85AF 8DFC0173;      1179          ASL.0+1->@-                   ..SECTOR + 1
85B3 F80073;        1180          00H->@-                       ..HEAD 0
85B6 9D73;          1181          ASL.1->@-                     ..TRACK
85B8 0F73;          1182          @PARA->@-                     ..HEAD AND UNIT
85BA F803A8;        1183          3->CMDCNT.0                   ..3 COMMAND BYTES IN SEEK
85BD 9D3AC4;        1184          ASL.1;BNZ SEEK5               ..IF TRACK IS 0,
85C0 28;            1185          DEC CMDCNT                    ..ONLY 2 COMMAND BYTES
85C1 F807;          1186          RCCMD                         ..AND RECAL COMMAND INSTEAD
85C3 C8;            1187          LSKP
85C4 F80F73;        1188 SEEK5    SKCMD->@-                     ..OF SEEK COMMAND
85C7 F80473;        1189          BC->@-                        ..BYTE COUNT
85CA F80057;        1190          DMANOP->@IOCBPTR              ..DMANOP
85CD D484CD;        1191          CALL CMD
85D0 D48511;        1192          CALL WAIT
85D3 F88FB7;        1193          A.1(STA0)->IOCBPTR.1          ..POINT AT RESULT STATUS
85D6 F810A7;        1194          A.0(STA0)->IOCBPTR.0
85D9 07FAC0;        1195          @IOCBPTR.AND.0C0H             ..NORMAL TERMINATION ?
85DC 32ED;          1196          BZ SEEK40                     ..EXIT IF YES
85DE 07FA10;        1197          @IOCBPTR.AND.10H              ..GET DRIVE FAIL BIT
85E1 FE52;          1198          *2->@SP                       ..LINE UP FOR STATUS
85E3 07FA08;        1199          @IOCBPTR.AND.08H              ..GET DRIVE INACTIVE BIT
85E6 FEFEFEF1;      1200          *2*2*2.OR.@                   ..COMBINE WITH ABOVE
85EA C6F802;        1201          LSNZ;02H                      ..IF NEITHER, GET BAD TERM BIT
85ED ADD5;          1202 SEEK40   ->ASL.0;EXIT                  ..LOAD RESULT STATUS, AND EXIT
85EF ;              1203
85EF ;              1204 ..*****************************************************************************
85EF ;              1205
85EF ;              1206          ..READ SECTOR ROUTINES  (ALL MUST BE PRECEDED BY SEEK OR SEEKA)
85EF ;              1207
85EF ;              1208          ..READTR READS USING DMA ADDRESS FROM @ PARA AS DESCRIBED IN SEEK.
85EF ;              1209          ..HIGH NIBBLE OF UNIT # INDICATES # OF SECTORS TO READ (@512B/S).
```

```
85EF ;              1210
85EF D4861D4603;    1211 READTR  CALL SETBC;DC RDCMD,DMAI
85F4 D5;            1212         EXIT
85F5 ;              1213
85F5 ;              1214         ..READST READS 1 SECTOR, USING DMA ADDRESS @ PARA
85F5 ;              1215
85F5 D486294603;    1216 READST  CALL SETRW;DC RDCMD,DMAI
85FA D5;            1217         EXIT
85FB ;              1218
85FB ;              1219         ..READA READS 1 SECTOR, DOES NOT SETUP DMA POINTER
85FB ;              1220
85FB D486344603;    1221 READA   CALL DORW;DC RDCMD,DMAI
8600 D5;            1222         EXIT
8601 ;              1223 ..*****************************************************************************
8601 ;              1224
8601 ;              1225         ..WRITE SECTOR ROUTINES (ALL MUST BE PRECEDED BY SEEK OR SEEKA)
8601 ;              1226         ..ALL DO CRC READ AFTER WRITE IF WRITE TERMINATED OKAY
8601 ;              1227
8601 ;              1228         ..WRITTR WRITES MULTIPLE SECTORS AS DESCRIBED FOR READTR
8601 ;              1229
8601 D4861D4502;    1230 WRITTR  CALL SETBC;DC WTCMD,DMAO
8606 3014;          1231         BR CKFCRC
8608 ;              1232
8608 ;              1233         ..WRITST WORKS LIKE READST
8608 ;              1234
8608 D486294502;    1235 WRITST  CALL SETRW;DC WTCMD,DMAO
860D 3014;          1236         BR CKFCRC
860F ;              1237
860F ;              1238         ..WRITA WORKS LIKE READA
860F ;              1239
860F D486344502;    1240 WRITA   CALL DORW;DC WTCMD,DMAO
8614 ;              1241
8614 ;              1242         ..COMMON CRC CHECK
8614 ;              1243
8614 8D3A1C;        1244 CKFCRC  ASL.O;BNZ WRITEX        ..IF TERMINATED OKAY
8617 D48634;        1245         CALL DORW
861A 4601;          1246         DC RDCMD,CRCREAD        ..DO READ CRC
861C D5;            1247 WRITEX  EXIT
861D ;              1248 ..*****************************************************************
861D ;              1249
861D ;              1250                 ..COMMON BYTE COUNT SETUP ENTER POINT
861D ;              1251
861D F801A7;        1252 SETBC   A.O(IOCB+1)->IOCBPTR.O
8620 F88FB7;        1253         A.1(IOCB+1)->IOCBPTR.1          ..POINT AT BC IN IOCB
8623 0FFAF0;        1254         @PARA.AND.OFOH                  ..RETRIEVE # OF SECTORS AND STUFF
8626 F6F657;        1255         /2/2->@IOCBPTR                  ..BC (# BYTES = 128 X BC)
8629 ;              1256
8629 ;              1257                 ..COMMON SETUP DMA POINTER ENTER POINT
8629 ;              1258
8629 1F1F1F;        1259 SETRW   INC PARA;INC PARA;INC PARA    ..POINT AT HI BUFFER ADDRESS BYTE
862C 4FB0;          1260         @PARA!->DMAPTR.1              ..SET UP DMA OUTPUT BUFFER POINTER
862E 0FA0;          1261         @PARA->DMAPTR.O
8630 2F2F;          1262         DEC PARA;DEC PARA             ..POINT BACK AT UNIT BYTE
8632 2F2F;          1263         DEC PARA;DEC PARA
8634 ;              1264
8634 ;              1265                 ..COMMON DO READ OR WRITE ROUTINE
8634 ;              1266
8634 ;              1267         ..CALL WITH IMMEDIATE BYTE = COMMAND, NEXT BYTE = DMA OPERATION
8634 ;              1268         ..CAN DO ANY 9 WORD COMMAND, LEAVES ASL.O WITH RESULT STATUS
8634 ;              1269         ..MUST HAVE SEEK DONE FIRST FOR ENTIRE SETUP
8634 ;              1270
8634 F802A7;        1271 DORW    A.O(IOCB+2)->IOCBPTR.O
8637 F88FB7;        1272         A.1(IOCB+2)->IOCBPTR.1        ..PT TO COMMAND IN IOCB
863A 4657;          1273         @LINK!->@IOCBPTR              ..AND LOAD IT
863C 2727;          1274         DEC IOCBPTR;DEC IOCBPTR       ..POINT TO DMA DIRECTION
863E 4657;          1275         @LINK!->@IOCBPTR              ..AND LOAD IT
8640 F809A8;        1276         9->CMDCNT.O                   ..9 OUTPUTS FOR COMMAND
8643 D484CD;        1277         CALL CMD                      ..GO DO IT
8646 D48511;        1278         CALL WAIT                     ..AND GET RESULTS
8649 F810A7;        1279         A.O(STAO)->IOCBPTR.O
864C F88FB7;        1280         A.1(STAO)->IOCBPTR.1          ..POINT TO STATUS O BYTE
864F 07FAC0;        1281         @IOCBPTR.AND.OCOH             ..IF NORMAL TERMINATION
8652 3278;          1282         BZ DORWEX                     ..GO EXIT
8654 07FA10;        1283         @IOCBPTR.AND.10H              ..PUT DRIVE FAIL (STAO BIT 4)
8657 FE52;          1284         *2->@SP                       ..INTO STATUS BIT 5
8659 47FA08;        1285         @IOCBPTR!.AND.08H             ..OR DR. INACTIVE (STAO BIT 3)
865C FEFEFEF152;    1286         *2*2*2.OR.@->@SP              ..INTO STATUS BIT 6
```

```
8661 07FA02;          1287          @IOCBPTR. AND. 02H                ..OR WRITE PROT.  (STA1 BIT 1)
8664 FEFEFEF152;      1288          *2*2*2. OR. @->@SP                ..INTO STATUS BIT 4
8669 47FA20;          1289          @IOCBPTR!. AND. 20H               ..OR CRC ERROR (STA1 BIT 5)
866C F6F6F152;        1290          /2/2. OR. @->@SP                  ..INTO STATUS BIT 3
8670 07FA40;          1291          @IOCBPTR. AND. 40H                ..OR FOUND DDM (STA2 BIT 6)
8673 FEF1;            1292          *2. OR. @                         ..INTO STATUS BIT 7
8675 C6F802;          1293          LSNZ;02H                          ..SET TERM. ERROR IF NONE ABOVE
8678 ADD5;            1294 DORWEX   ->ASL. 0;EXIT                     ..STORE STATUS BYTE, EXIT
867A ;                1295
867A ;                1296
867A ;                1297 ..*****************************************************************************
867A ;                1298 ..              KEYBOARD READ / WRITE ROUTINE
867A ;                1299 ..
867A ;                1300 ..       USES SEEKA, WRITA, READA, RECAL, ENDSTRG, CKSTRG
867A ;                1301 ..
867A 8DC8;            1302 WDISK    ASL. 0;LSKP                       ..(ASL WAS ZEROED)
867C F80173;          1303 RDISK    1->@-                             ..STORE READ / WRITE FLAG
867F D48499;          1304          CALL SPECIFY                      ..INITIALIZE 765
8682 D4873241;        1305          CALL CKSTRG;DC 'A'                ..PROMPT FOR ADDRESS
8686 8DA0;            1306          ASL. 0->DMAPTR. 0
8688 9DB0;            1307          ASL. 1->DMAPTR. 1                 ..STORE DESTINATION ADDRESS
868A D4873244;        1308          CALL CKSTRG;DC 'D'                ..PROMPT FOR DRIVE #
868E 8D52;            1309          ASL. 0->@SP                       ..STORE FOR RECAL AND SEEK
8690 FF0433DF;        1310          -4;BDF WREREX                     ..ERROR IF DRIVE # TOO HIGH
8694 82AF;            1311          SP. 0->PARA. 0
8696 92BF;            1312          SP. 1->PARA. 1                    ..POINT PARA @ DRIVE #
8698 22;              1313          DEC SP
8699 F800BD;          1314          0->ASL. 1                         ..SET FOR TRACK 0
869C D4859C;          1315          CALL SEEKA                        ..DO SEEK (DOES RECAL)
869F 8D3AD5;          1316          ASL. 0;BNZ DERROR                 ..EXIT IF TERM ERROR
86A2 D4873254;        1317          CALL CKSTRG;DC 'T'                ..PROMPT FOR TRACK #
86A6 8D73;            1318          ASL. 0->@-                        ..STORE FOR SEEK LATER
86A8 FF4633DF;        1319          -MAXTRK;BDF WREREX                ..ERROR IF TRACK # TOO HIGH
86AC D48735;          1320          CALL ENDSTRG                      ..PROMPT FOR SECTOR #
86AF 53;              1321          DC 'S'                            ..LAST PROMPT, END INPUTS WITH CR
86B0 2D;              1322          DEC ASL                           ..SEEK WANTS SECTOR # -1
86B1 8DFF0933DF;      1323          ASL. 0-MAXSEC;BDF WREREX          ..ERROR IF SECTOR # TOO HIGH
86B6 1202BD;          1324          INC SP;@SP->ASL. 1                ..RETRIEVE TRACK #
86B9 82AF;            1325          SP. 0->PARA. 0
86BB 92BF;            1326          SP. 1->PARA. 1
86BD 1F;              1327          INC PARA                          ..POINT PARA @ DRIVE # AGAIN
86BE D4859C;          1328          CALL SEEKA                        ..AND SEEK (SETS UP FOR READ/WRITE)
86C1 8D3AD5;          1329          ASL. 0;BNZ DERROR                 ..EXIT IF TERM ERROR
86C4 1212;            1330          INC SP;INC SP
86C6 0232CE;          1331          @SP;BZ CALWRT                     ..IF READ FLAG SET,
86C9 D485FB;          1332          CALL READA                        ..DO IT
86CC 30D1;            1333          BR CKRDWR
86CE D4860F;          1334 CALWRT   CALL WRITA                        ..OTHERWISE DO WRITE
86D1 8DC282AD;        1335 CKRDWR   ASL. 0;LBZ RENTER                 ..SUCCESSFUL EXIT IF TERM OKAY
86D5 D483F0;          1336 DERROR   CALL OSTRNG
86D8 0D0A4449534B;    1337          DC CR,LF,'DISK',0                 ..OTHERWISE PRINT DISK ERROR
86DE 00;
86DF C08085;          1338 WREREX   LBR ERROR
86E2 ;                1339 ..*****************************************************************************
86E2 ;                1340 ..                    KEYBOARD INPUT/OUTPUT ROUTINE
86E2 ;                1341 ..
86E2 ;                1342 ..       USES CKSTRG,ENDSTRG, USES AUX. 0 FOR TEMP STORE
86E2 ;                1343 ..
86E2 ;                1344 ..              OUTPUT ROUTINE
86E2 ;                1345 ..
86E2 D4873247;        1346 OUTPORT  CALL CKSTRG;DC 'G'                ..PROMPT FOR GROUP #
86E6 8D73;            1347          ASL. 0->@-                        ..STORE FOR LATER
86E8 D4873250;        1348          CALL CKSTRG;DC 'P'                ..PROMPT FOR PORT #
86EC 8DFA07C28085;    1349          ASL. 0. AND. 07H;LBZ ERROR        ..GET PORT #, ERROR IF 0
86F2 F960AE;          1350          . OR. 60H->AUX. 0                 ..MAKE 6X INSTR., STORE
86F5 D4873542;        1351          CALL ENDSTRG;DC 'B'               ..PROMPT FOR OUTPUT BYTE
86F9 12;              1352          INC SP                            ..POINT SP @ GROUP #
86FA 6122;            1353          OUT DISKSEL;DEC SP                ..OUTPUT IT, FIX STACK
86FC F8D373;          1354          0D3H->@-                          ..PUT RETURN (SEP R3) ON STACK
86FF 8D73;            1355          ASL. 0->@-                        ..PUT OUTPUT BYTE ON STACK
8701 8E52;            1356          AUX. 0->@SP                       ..OUTPUT INSTRUCT.  ON STACK
8703 D2;              1357          SEP SP                            ..DO OUTPUT COMMAND
8704 C082AD;          1358          LBR RENTER                        ..BACK TO UTILITY (FIXES SP, GROUP #)
8707 ;                1359 ..
8707 ;                1360 ..       INPUT ROUTINE
8707 ;                1361 ..
8707 D4873247;        1362 INPORT   CALL CKSTRG;DC 'G'                ..PROMPT FOR GROUP #
```

```
870B 8D73;          1363          ASL. O->@-                          ..STORE FOR LATER
870D D4873550;      1364          CALL ENDSTRG;DC 'P'                 ..PROMPT FOR PORT #.
8711 8DFA07C28085;  1365          ASL. O. AND. 07H;LBZ ERROR          ..ERROR IF INPUT PORT = O
8717 F96852;        1366          .OR. 68H->@SP                       ..MAKE INPUT INST., STORE
871A 12;            1367          INC SP                              ..POINT SP @ GROUP #
871B 6122;          1368          OUT BDSEL;DEC SP                    ..OUTPUT IT, FIX SP
871D F8D373;        1369          0D3H->@-                            ..PUT RETURN (SEP R3) ON STACK
8720 E3;            1370          SEX PC                              ..(THIS SO INPUT WON'T CLOBBER SP)
8721 D2;            1371          SEP SP                              ..DO INPUT
8722 22;            1372          DEC SP                              ..FIX SP
8723 AE;            1373          ->AUX. O                            ..STORE INPUT FOR LATER
8724 6101;          1374          OUT BDSEL;DC TRMINL                 ..SEL. TERMINAL
8726 D481A20A;      1375          CALL TYPE6;DC LF
872A 8EBF;          1376          AUX. O->CHAR. 1
872C D481AE;        1377          CALL TYPE2                          ..AND DISPLAY BYTE
872F C082AD;        1378          LBR RENTER
8732 ;              1379
8732 ;              1380 ..*****************************************************************************
8732 ;              1381 ..                      HEX STRING INPUT ROUTINE
8732 ;              1382 ..
8732 ;              1383 ..       TYPES PROMPT (FROM @ LK!) =
8732 ;              1384 ..       ZEROES ASL, INPUTS CHARECTERS UNTIL NON-HEX INPUT
8732 ;              1385 ..       CKSTRG GOES TO ERROR IF NOT SPACE
8732 ;              1386 ..       ENDSTRG GOES TO ERROR IF NOT CR
8732 ;              1387 ..
8732 F820C8;        1388 CKSTRG   ' ';LSKP                            ..LOAD SPACE
8735 F80D73;        1389 ENDSTRG  CR->@-                              ..OR CR FOR CHECK
8738 46BF;          1390          @LINK!->CHAR. 1
873A D481A4;        1391          CALL TYPE                           ..TYPE PROMPT
873D D481A23D;      1392          CALL TYPE6;DC '='
8741 F800ADBD;      1393          O->ASL. O, ASL. 1                   ..ZERO INPUT  REGISTER
8745 D482F0;        1394          CALL READHX                         ..INPUT UNTIL 1ST NON HEX
8748 12;            1395          INC SP
8749 F3;            1396          .XOR. @                             ..IF DOESN'T MATCH ABOVE
874A CA8085;        1397          LBNZ ERROR                          ..ABORT
874D D5;            1398 CKSTEX   EXIT
874E ;              1399 ..
874E ;              1400 ..*****************************************************************************
874E ;              1401 ..                    MONITOR SELF TEST ROUTINE
874E ;              1402 ..
874E ;              1403 ..       DOES CHECKSUM OF PROM, WRITES AND READS 55 AND AA TO ALL RAM
874E ;              1404 ..
874E F800A752;      1405 TEST     A. O(UT71)->IOCBPTR. O, @SP         ..ZERO CHECKSUM,
8752 F880B7;        1406          A. 1(UT71)->IOCBPTR. 1              ..PT TO START OF PROM
8755 47F452;        1407 TROM1    @IOCBPTR!+@->@SP                    ..ADD ALL PROM TOGETHER
8758 97FF88;        1408          IOCBPTR. 1-A. 1(RAMADR)
875B 3B55;          1409          BM TROM1                            ..LOOP UNTIL IT HITS RAM
875D 02326F;        1410          @SP;BZ RAMTEST                      ..IF CHECK NOT = O
8760 D483F0;        1411          CALL OSTRNG
8763 0A524F4D20;    1412          DC LF,'ROM BAD',O                   ..PRINT BAD ROM
8768 42414400;
876C C082AD;        1413          LBR RENTER                          ..EXIT TO UTILITY
876F ;              1414 ..
876F F880B7;        1415 RAMTEST  A. 1(UT71)->IOCBPTR. 1
8772 27;            1416          DEC IOCBPTR                         ..POINT JUST BELOW PROM
8773 E7;            1417          SEX IOCBPTR
8774 F85573;        1418 RAM1     55H->@-
8777 97FF87;        1419          IOCBPTR. 1-A. 1(RAMADR-1)
877A 3A74;          1420          BNZ RAM1                            ..FILL ALL RAM WITH 55
877C 17;            1421          INC IOCBPTR                         ..PT AT JUST ABOVE PROM
877D F0FBFF57;      1422 RAM2     @. XOR. OFFH->@IOCBPTR              ..MAKE 55 -> AA AND RESTORE
8781 72FBAA;        1423          @!. XOR. OAAH
8784 327D;          1424          BZ RAM2                             ..CHECK AA WROTE
8786 27;            1425          DEC IOCBPTR
8787 97FF80;        1426          IOCBPTR. 1-A. 1(UT71)               ..IF FIRST FAILURE IS IN PROM
878A 3A9D;          1427          BNZ BRAMM
878C D483F0;        1428          CALL OSTRNG                         ..PRINT OKAY
878F 0A4D454D4F;    1429          DC LF,'MEMORY OK',O
8794 5259204F4B00;
879A C082AD;        1430          LBR RENTER                          ..AND EXIT
879D ;              1431 ..
879D D483F0;        1432 BRAMM    CALL OSTRNG                         ..BAD RAM MESSAGE
87A0 0A52414D20;    1433          DC LF,'RAM BAD, P',O
87A5 4241442C2050;
87AB 00;
87AC 97BF;          1434          IOCBPTR. 1->CHAR. 1
87AE D481AE;        1435          CALL TYPE2                          ..PRINT PAGE #
```

```
87B1 C082AD;        1436          LBR RENTER                          ..AND EXIT
87B4 ;              1437 ..*************************************************************************
87B4 ;              1438
87B4 ;              1439        ..DISK I/O ENTRY TABLE AND ROM TEST CHECK BYTE
87B4 ;              1440
87B4 ;              1441                ORG UT71+007D8H
87D8 ;              1442
87D8 C0814F;        1443 CFRETS  LBR CFRET        ..COMMAND FILE RETURN POINT
87DB C085EF;        1444 READTRS LBR READTR       ..READ MULTIPLE SECTOR, SET DMA FROM PARA.
87DE C08601;        1445 WRITTRS LBR WRITTR       ..WRITE AS ABOVE
87E1 C085FB;        1446 READAS  LBR READA        ..READ SECTOR USING ALREADY SET DMA POINTER
87E4 C0860F;        1447 WRITAS  LBR WRITA        ..WRITE SECTOR AS ABOVE
87E7 C084B6;        1448 RECALS  LBR RECAL        ..RECALIBRATE, USE DMAPTR.0 FOR DR. #
87EA C0859C;        1449 SEEKAS  LBR SEEKA        ..SEEK BY TRACK & SECTOR IN ASL, @PARA = DR. #
87ED C084CD;        1450 CMDS    LBR CMD          ..OUTPUT COMMAND BYTES
87F0 C082AD;        1451 RENTERS LBR RENTER       ..UT71 RENTRY ADDRESS
87F3 C08511;        1452 WAITS   LBR WAIT         ..SERVICE FDC AFTER COMMAND
87F6 C08570;        1453 SEEKS   LBR SEEKST       ..SEEK BY PSN IN PARA. BLOCK
87F9 C085F5;        1454 READS   LBR READST       ..READ SECTOR, SET DMA FROM PARA. BLOCK
87FC C08608;        1455 WRITS   LBR WRITST       ..WRITE SECTOR AS ABOVE
87FF FF;            1456 CHECK   DC OFFH          ..CHECK SUM BYTE (SET AFTER ASSEMBLY)
8800 ;              1457
8800 ;              1458         END
0000
```

C R O S S R E F E R E N C E L I S T I N G

SYMBOL	ADDR	DEF	REFERENCES							
ADDOUT	80AF	0236	0229							
ADRPTR	0008		0042							
ASL	000D		0049	0151	0151	0225	0226	0252	0256	0374
			0376	0378	0379	0382	0477	0478	0479	0479
			0488	0490	0493	0494	0496	0497	0504	0505
			0528	0602	0603	0697	0697	0828	0828	0833
			0834	0840	0955	0976	0978	0979	0981	0986
			1156	1158	1162	1165	1166	1166	1179	1181
			1184	1202	1244	1294	1302	1306	1307	1309
			1314	1316	1318	1322	1323	1324	1329	1335
			1347	1349	1355	1363	1365	1393	1393	
AUX	000E		0051	0119	0309	0319	0347	0355	1015	1157
			1163	1165	1350	1356	1373	1376		
BC	0004		0928	1189						
BDSEL	0001		0073	0303	0861	1368	1374			
BEGIN	81C3	0438	0439	0460						
BOOT	8400	0953	0181	0953						
BRAMM	879D	1432	1427							
BS	0008		0065							
BYTCNT	8217	0487	0499							
BYTECNT	0007		0914	1074						
CAL	918C		0900	1009	1012					
CALL	0004		0036	0141	0202	0223	0236	0240	0246	0248
			0254	0263	0266	0268	0475	0484	0502	0519
			0521	0597	0626	0631	0635	0639	0645	0655
			0670	0682	0684	0693	0698	0708	0723	0810
			0811	1009	1012					
CALLR	8364	0770	0810	0811						
CALWRT	86CE	1334	1331							
CFREAD	8FFD		0030	0348						
CFRET	814F	0355	1443							
CFRETS	87D8	1443								
CHAR	000F		0052	0144	0228	0245	0262	0265	0316	0339
			0345	0353	0359	0364	0367	0420	0425	0436
			0444	0444	0452	0480	0644	0721	0788	1376
			1390	1434						
CHARAC	0002		0083	0352	0360	0441				
CHECK	87FF	1456								
CKDEC	812F	0332	0368							
CKFCRC	8614	1244	1231	1236						
CKHEX	83FC	0875								
CKHXE	815E	0367	0875							
CKRDWR	86D1	1335	1333							
CKSTEX	874D	1398								
CKSTRG	8732	1388	1305	1308	1317	1346	1348	1362		
CMD	84CD	1060	1038	1053	1191	1277	1450			
CMD1	84DA	1066	1062	1066						
CMD2	84D2	1063	1063	1069						
CMD3	84E4	1069	1067							
CMD4	84D8	1065	1068							
CMD5	84F0	1076	1076							
CMD6	84EF	1075	1080							
CMDCNT	0008		0887	1037	1052	1079	1080	1183	1185	1276
CMDS	87ED	1450								
CNT	000A		0043	0488	0490	0504	0505	0507	0527	0529
			0531	0558	0560	0564	0567	0569	0571	0573
			0580	0582	0585	0647	0649	0652		
CNTIN	8231	0502	0481							
COMCHK	83DB	0850	0845							

```
COMMA           002C            0063
COMMAND         0005            0912  1078  1112  1128
CR              000D            0067  0238  0311  0312  0599  0628  0658  0700
                                0710  0843  0844  1337  1389
CRCREAD         0001            0939  1131  1246
CRLF            0D0A            0070  0129  0204
CTLWRD          001D            0082  0306
DATA            0005            0913  1064  1068  1115  1119  1136
DATOUT          82D2    0644    0663
DECODE          80A0    0228    0258
DELAY           000C            0046  0243  0260  0299  0301  0718  0719  0808
                                0809
DELAY1          80EF    0285    0300  0808  0809
DELAY2          80F0    0287    0289
DERROR          86D5    1336    1316  1329
DEST            000D            0050  0546  0549  0553  0555  0557  0563  0572
                                0572  0574  0574  0579  0585
DEXIT           80EE    0283    0288
DIRECT          8261    0552    0547
DISKSEL         0001            0908  1061  1082  1099  1137  1353
DISPLY          82BD    0626    0165
DMA             0000            0937  1032
DMAI            0003            0941  1211  1216  1221
DMANOP          0000            0938  1036  1051  1132  1190
DMAO            0002            0940  1230  1235  1240
DMAPTR          0000            0884  0957  0957  1048  1260  1261  1306  1307
DMASEL          0004            0909  1073  1131  1132
DORW            8634    1271    1221  1240  1245
DORWEX          8678    1294    1282
DTL             00FF            0932  1175
ECHOTP          910A            0902  1011
ECHOTST         8109    0309
ENDSTRG         8735    1389    1320  1351  1364
ENDWAIT         856D    1137    1106  1127  1135
ENTER1          8389    0805    0872
ENTER2          838F    0808    0616  0803  0873
ENTRY           9040            0903  1016
EOF             0013            0068
EOT             0009            0930  1177
ERR1            8360    0757    0696  0701  0832  0839
ERRGO           828E    0577
ERROR           8085    0200    0148  0234  0600  0629  0686  0757  0956  1338
                                1349  1365  1397
EXITC           8363    0768    0777
EXITDF          8348    0748    0738  0745
EXITEF          834C    0750    0740
EXITM           832B    0727    0722
EXITOK          823F    0508    0491  0506
EXITR           8373    0781    0790
FILL            8240    0519    0171
FM              0040            0933
FND             8169    0373    0335
GETDTA          80C5    0254
GOUT71          83F9    0874
GPL3            001B            0931  1176
HEX1            81DA    0452    0448
FILE: UT70.XRF          DISK: UT70 WORK DISK I    R.H. ISHAM

HEX2            81E3    0457    0454
HEX3            81E5    0459    0450
HLT             003C            0935  1032
HUT             000F            0936  1033
INIT            8381    0801    0114
```

```
INIT1           83F3    0872
INIT2           83F6    0873
INPORT          8707    1362    0189
INSERT          83A7    0828    0167  0848
INSERT1         83AB    0829    0830
INTPC           0001            0885
INVCMD          0000            0924  1128
INVERT          8223    0493
IOCB            8F00            0894  0982  0983  1029  1030  1045  1046  1070
                                1071  1121  1122  1172  1173  1252  1253  1271
                                1272
IOCBPTR         0007            0886  0982  0983  0984  0992  0993  0994  0995
                                0997  1011  1014  1015  1029  1030  1031  1045
                                1046  1047  1050  1070  1071  1072  1077  1097
                                1101  1101  1103  1104  1105  1108  1121  1122
                                1161  1162  1165  1168  1169  1172  1173  1174
                                1190  1193  1194  1195  1197  1199  1252  1253
                                1255  1271  1272  1273  1274  1274  1275  1279
                                1280  1281  1283  1285  1287  1289  1291  1405
                                1406  1407  1408  1415  1416  1417  1419  1421
                                1422  1425  1426  1434
LF              000A            0066  0228  0311  0312  0316  0421  0633  0847
                                0853  1022  1337  1375  1412  1429  1433
LINES           0014            0075
LINK            0006            0038  0155  0157  0721  0771  0772  0773  0774
                                0775  0776  0783  0784  0786  0787  1273  1275
                                1390
LNECNT          000F            0074  0653
LOAD            8405    0954    0179
LOAD1           842F    0975    0961
LOAD2           8435    0978    0987
LOADOK          845E    1005
LOOP            800C    0103    0112
MAXSEC          0009            0945  1323
MAXTRK          0046            0944  1319
MICRO           A843            0904  0992  0993
MICTST          844F    0992
MOVDN           826B    0557    0565
MOVDN1          8273    0563    0559
MOVE            82F7    0682    0169
MOVUP           8278    0567    0556
MSGE            831C    0718    0871
MSGE1           8322    0721    0725
N               0002            0929  1178
NEC             0008            0943  1061  1099
NECSTA          0004            0911  1063  1066  1076  1109  1117  1130  1134
NEXT            815B    0364    0356
NFND            8137    0337    0333  0370
NOECHO          812A    0319    0317
NOLOAD          847E    1021    0976  0981  0986  1000
NOTDON          82DD    0652    0648
NTDATA          83CA    0843    0837
NULL            0000            0062
NXCHAR          81CA    0444
NXTCEL          8249    0526    0530  0532
FILE: UT70.XRF          DISK: UT70 WORK DISK I    R.H. ISHAM

NXTCHR          83BB    0836    0841  0851  0854
OLDDTA          80B7    0243    0233
OPTION          8200    0475    0627  0694
OSTRNG          83F0    0871    0128  0203  0237  0249  0269  0310  0632  0640
                                0656  1005  1021  1336  1411  1428  1432
OUT1            80CD    0260    0241  0636
```

```
OUTPORT          86E2    1346    0187
OUTPUT           82C5    0631    0660
PARA             000F            0889  0954  0954  1157  1157  1158  1159  1159
                                 1182  1254  1259  1259  1259  1260  1261  1262
                                 1262  1263  1263  1311  1312  1325  1326  1327
PC               0003            0035  0146  0283  0285  0299  0302  0344  0614
                                 0615  0768  0773  0774  0775  0776  0781  0783
                                 0784  0801  0802  0805  0806  0815  0860  1060
                                 1081  1098  1111  1125  1370
PGMSRT           0005            0076  0805  0806
PRINT1           8336    0741    0747
PRMPT            8034    0127    0614  0615
PRMPT1           8096    0207    0231
PRMPT2           8246    0524    0483  0650  0687
PRMPT5           835D    0756    0843
PRNTRF           832C    0737    1090
PROMPT           002A            0077  0130  0313
PTER             000C            0047
PTR              000C            0048
RAM1             8774    1418    1420
RAM2             877D    1422    1424
RAMADR           8800            0893  1408  1419
RAMTEST          876F    1415    1410
RCA              0001            0942  1082  1137
RCCMD            0007            0919  1049  1186
RDCMD            0046            0921  1211  1216  1221  1246
RDISK            867C    1303    0185
RDWAIT           8153    0357    0358
READ             813E    0344    0142  0315
READ1            8145    0350    0351  0354
READA            85FB    1221    0985  1332  1446
READAD           8303    0693    0520  0683
READAH           813B    0342    0671  0709  0829  0836  0838  0953
READAS           87E1    1446
READCR           8314    0708    0711  0846  0852
READHX           82F0    0670    0224  0255  0476  0485  0503  0598  0672  0699
                                 0831  1394
READS            87F9    1454
READST           85F5    1216    1454
READTR           85EF    1211    1444
READTRS          87DB    1444
RECAL            84B6    1045    1448
RECALS           87E7    1448
RENTER           82AD    0611    0207  0524  0756  0874  1023  1335  1358  1378
                                 1413  1430  1436  1451
RENTERS          87F0    1451
RENTR1           82B4    0614    0611  0612
RET              919C            0901  1010
RETN             0005            0037  0810  0812  1010  1013
RETR             8374    0783    0812
REXIT            8139    0339    0365  0384
ROWLEN           0028            0078
RUN              829F    0597    0175
RUN1             82A7    0602
FILE: UT70.XRF          DISK: UT70 WORK DISK I    R.H. ISHAM

SAMELN           82EB    0662    0654
SCAN             8048    0147    0150
SCAN1            8041    0144    0120
SCNLTR           803E    0141
SEEK40           85ED    1202    1196
SEEK5            85C4    1188    1184
SEEKA            859C    1172    0975  0980  1315  1328  1449
```

```
SEEKAS          87EA    1449
SEEKS           87F6    1453
SEEKST          8570    1156    1453
SEMCOL          003B            0064
SETBC           861D    1252    1211    1230
SETRW           8629    1259    1216    1235
SHRES           858E    1166    1164
SISCMD          0008            0923    1064    1112
SKCMD           000F            0920    1188
SP              0002            0034    0144    0153    0153    0200    0201    0307    0359
                                0361    0375    0381    0440    0442    0487    0489    0493
                                0495    0496    0498    0544    0545    0548    0552    0554
                                0567    0569    0571    0573    0770    0785    0785    0813
                                0814    0815    0954    0954    1007    1008    1075    1115
                                1123    1126    1160    1163    1167    1170    1198    1284
                                1286    1288    1290    1309    1311    1312    1313    1324
                                1324    1325    1326    1330    1330    1331    1352    1353
                                1356    1357    1366    1367    1368    1371    1372    1395
                                1405    1407    1410
SPACE           0020            0069    0270    0480    0641    0695
SPCMD           0003            0918    1034
SPCOUT          82CD    0639    0664
SPECIFY         8499    1029    0960    1304
SRC             000B            0044    0225    0226    0245    0252    0256    0257    0262
                                0265    0477    0478    0487    0489    0494    0495    0497
                                0498    0526    0528    0545    0548    0552    0554    0557
                                0563    0568    0568    0570    0570    0579    0585    0644
                                0653    0833    0834    0840    0840
SRT             0010            0934    1033
STA0            8F10            0895    1103    1104    1193    1194    1279    1280
START           802C    0118    0801    0802
STATUS          0003            0084    0350    0357    0438
STK             BFFF            0899    1007    1008
SUBLP           8583    1162    1169
SUBST           8099    0223    0173
TAB2            805D    0164    0145
TERMCNT         0004            0910
TEST            874E    1405    0177
TEXIT           819F    0412    0445    0447
TIMALC          80FE    0299    0118
TKTABL          8424    0971    0958    0959
TMPRG1          0007            0039
TMPRG2          0008            0040
TMPRG3          0009            0041
TOPSTK          8CFF            0057    0200    0201    0813    0814
TPOFF           83E8    0860    0127
TPTR            000B            0045    0145    0146    0147    0147    0148    0155    0157
TRKCNT          0009            0888    0958    0959    0977    0979    0984    0987
TRMINL          0001            0079    0862    1374
TROM1           8755    1407    1409
TY1             81B9    0430    0427
TY2             81C0    0434    0422
TY3             81C2    0436    0424    0432
FILE: UT70.XRF          DISK: UT70 WORK DISK I    R.H. ISHAM

TYPE            81A4    0420    0405    1391
TYPE2           81AE    0425    0247    0264    0267    0646    1377    1435
TYPE5           81A0    0414    0408
TYPE5D          819C    0408
TYPE6           81A2    0417    0152    0847    0853    1375    1392
TYPED           8198    0405    0724
UARTBD          0001            0080    0304
UP              8290    0579    0575    0586
```

UP1	8298	0585	0581							
URTCTL	0003		0081	0305						
USRBYE	829D	0588	0550	0561	0577	0583				
USRFIL	824B	0528	0522							
USRMOV	8254	0544	0685	0966						
UT71	8000		0056	0087	0091	0116	0198	0281	0297	0330
			0403	0406	0410	0473	0680	0754	0764	0869
			0893	0951	0964	1088	1405	1406	1415	1426
			1441							
UT71A	8026	0114	0109							
WAIT	8511	1097	1124	1192	1278	1452				
WAIT1	851D	1103	1100							
WAIT10	8565	1134	1134							
WAIT2	8514	1098	1102							
WAIT3	8529	1109	1109							
WAIT4	8537	1115	1110							
WAIT5	8528	1108	1113							
WAIT6	853B	1117	1117							
WAIT7	8545	1121	1118							
WAIT8	8539	1116	1119							
WAIT9	855B	1130	1130							
WAITS	87F3	1452								
WDISK	867A	1302	0183							
WRAM	8C1F		0058	0100	0101					
WREREX	86DF	1338	1310	1319	1323					
WRITA	860F	1240	1334	1447						
WRITAS	87E4	1447								
WRITEX	861C	1247	1244							
WRITS	87FC	1455								
WRITST	8608	1235	1455							
WRITTR	8601	1230	1445							
WRITTRS	87DE	1445								
WTCMD	0045		0922	1230	1235	1240				

Appendix H
ASCII - Hex Table

LEAST SIGNIFICANT HEX DIGIT	MOST SIGNIFICANT HEX DIGIT							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	—	=	M	]	m	}
E	SO	RS	.	>	N	↑	n	~
F	SI	US	/	?	O	←	o	DEL

NOTES:

(1) Parity bit in most significant hex digit not included.

(2) Characters in columns 0 and 1 (as well as SP and DEL) are non-printing.

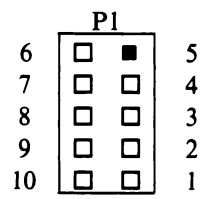
(3) Model 33 Teletypewriter prints codes in columns 6 and 7 as if they were column 4 and 5 codes.

92CS-34736

Appendix I—
Connection List for Terminal Interface Cable

CDP 185516
EIA R5232C Terminal

P1	P2	Signal
1	1	Ground
2	2	Data to MS2000
3	3	Data to Terminal
10	7	Signal Ground
7	5	Clear to Send
6	6,8	Data Set Ready— Held High by MS2000



Note: P2 is a 25—pin D connector, male.
(Adaptor supplied to convert to female.)

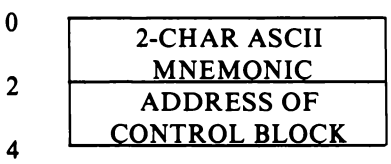
Appendix J — Adding Generic Devices

Three tables are used when generic devices are added to the RCA Microdisk Development System. These tables are:

I. Generic Device Table
II. Control Block Table
III. Device Driver Table

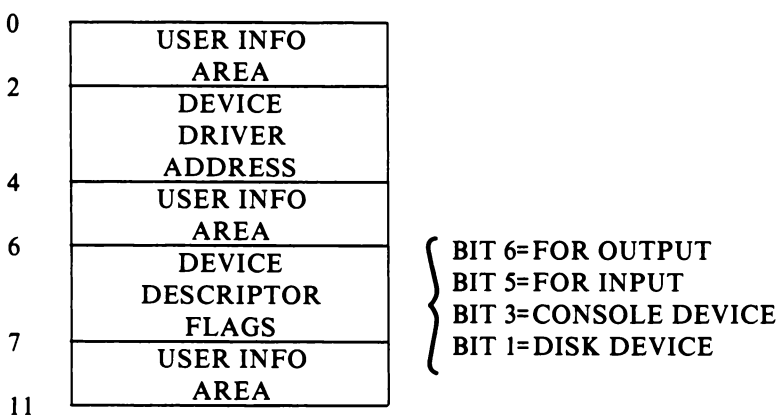
The Generic Device Table contains the two-character mnemonic for the added device and a pointer to the control block, the Control Block Table, the Device Descriptor Flags, and the unused area for user information. The Device Driver Table contains three long branches to routines that control the turning on and off of the device and the character input or output instructions. Only the Generic Device Table entry for the added device must be in a specific place. Only three devices may be added to the system.

I. Generic Table Entry



Note: a zero must be placed after the last entry to terminate the Generic Device Table.

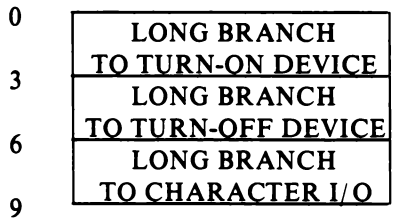
II. Control Block Entry



Notes: Bit 5 in the first byte of the IOCB must be set to zero before the IOCB is opened for added generic devices.

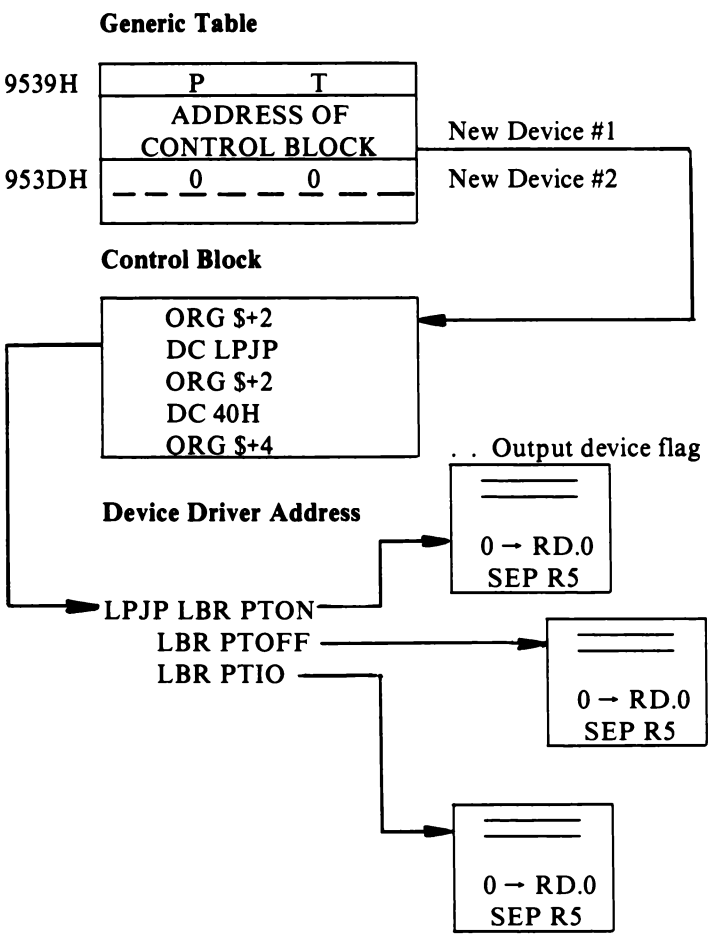
Register F must have the same value exiting these routines as when the routine was entered.

III. Device Driver Table Entry



An example follows for adding to the system a line printer with the mnemonic PT

Generic Table



Appendix K
MicroDisk Development System MS2000 Specifications

System Components

20-slot Industrial Microboard Chassis
CDP18S605 Microboard Computer less memory
CDP18S618 Microboard Memory configured as 32-kilobyte RAM
CDP18S628 Microboard Memory configured as 30 kilobyte RAM plus 2-kilobyte ROM
CDP18S651 Microboard Disk Controller
MSIM 50 Dual Microfloppy Disk Drive Module
MSIM 40 Power Supply
UT71 Monitor Software, ROM-based (On CDP18S628)
CDP18S516 EIA RS232C Terminal Interface Cable

Memory

RAM
32 kilobytes at 0000H - 7FFFH
30 kilobytes at 8800H - FFFFH

ROM
2 Kilobytes UT71 at 8000H - 87FFH

Disk Drive and Controller

Dual Microfloppy Disk Drive Module MSIM 50
Occupies 8 Microboard slots
Capacity: 322.5 kilobytes per drive
Tracks: 70
Sectors: 9 per track, 512 bytes per sector
Transfer rate: single density 250 kilobits per second
double density 500 kilobits per second
Step rate time: 15 ms
Step settling time: 15 ms
Head load time: 60 ms
Latency: 50 ms (average)
Rotational speed: 600 rpm
Power requirements: +15 V at 800 mA typ. operating
+5 V at 850 mA typ. operating
Signal cable: 26-line to connector on Microboard Disk Controller CDP18S651

Power Supply and Controls

Plug-in Power Supply
Output:
+5 V at 3 A
+15 V at 1.6 A, 2A peak
-15 V at 0.8 A
Input:
90 to 132 V, 47 to 440 Hz (MS2000)
180 to 264 V, 47 to 440 Hz (MS2000E)
Fuse: 1A, slow-blow, front-panel mounted
Controls:
Power on-off switch - front panel
RESET - RUN U switch
RESET - RUN P switch
Indicators:
RUN LED
+5 V ON LED

Dimensions

Height: 5.76 inches (146 mm)
Width: 14.7 inches (373 mm)
Depth: 10.08 inches (256 mm)

Weight: 18.5 pounds (8.4 kilograms)

Operating Temperature Range

5°C to 40°C

Expansion Capabilities

Four standard Microboard slots available in chassis
Reserve power available:
+5 V - 1 A
+15 V - 500 mA
-15 V - 800 mA

Text Editor Commands

Move Pointer
Delete
Append
Insert
Find
Save
Search & Substitute
Type
Output

Monitor Program Commands

Monitor Self Test
Read or Modify Memory
Read Saved State of CPU Registers
Start Program at Given Location
Load MicroDOS Operating System
Move Memory
Fill Memory
Substitute Memory

MicroDOS Operating System Commands

List Directory
List Free Space on Disk
Copy Disk File to Terminal, Line Printer, or another File
Delete File Name
Rename File
Convert ASCII-Hex Object File to Binary
Format a New Disk
Verify Disk Files
Merge Files
Save Memory under File Name
Examine Disk File Contents
Organize Disk Files
Transfer Files from PERTEC Unit/Track Format to MicroDOS
Transfer Files from Cassette Tape to MicroDOS
Translate CRA or ASM4 Assembly Language Source Code into ASM8 File

Appendix L
Contents Directory of MS2000 System Diskette (Typical)

DRIVE: 1 DISKID: 12/9/83 MICRODISK 1.0 (c) 1982 RCA CORPORATION

ASM8	.CM	ATTR WDSC.1	SSN 00058	SIZE 00025	DEN 30
CDSBIN	.CM	ATTR WDSC.1	SSN 00229	SIZE 00004	DEN 61
CDSBIN	.SR	ATTR WD...2	SSN 00083	SIZE 00033	DEN 31
CONASM	.CM	ATTR WDSC.1	SSN 00185	SIZE 00020	DEN 41
COPY	.CM	ATTR WDSC.1	SSN 00225	SIZE 00004	DEN 60
DEL	.CM	ATTR WDSC.1	SSN 00278	SIZE 00004	DEN 71
DIAG	.CM	ATTR WDSC.1	SSN 00284	SIZE 00004	DEN 73
DIR	.CM	ATTR WDSC.1	SSN 00036	SIZE 00014	DEN 10
EDIT	.CM	ATTR WDSC.1	SSN 00310	SIZE 00013	DEN 84
EXAM	.CM	ATTR WDSC.1	SSN 00212	SIZE 00011	DEN 50
FORMAT	.CM	ATTR WDSC.1	SSN 00208	SIZE 00004	DEN 43
FREE	.CM	ATTR WDSC.1	SSN 00205	SIZE 00003	DEN 42
HELP	.CM	ATTR WDSC.1	SSN 00116	SIZE 00003	DEN 32
HELP	.MSG	ATTR WDS. .2	SSN 00119	SIZE 00028	DEN 33
MEM	.CM	ATTR WDSC.1	SSN 00050	SIZE 00003	DEN 11
MEM	.SR	ATTR WD...2	SSN 00233	SIZE 00022	DEN 62
MEMTST	.CM	ATTR WDSC.1	SSN 00223	SIZE 00002	DEN 51
MERGE	.CM	ATTR WDSC.1	SSN 00147	SIZE 00014	DEN 34
OP	.SYS	ATTR WDSC.3	SSN 00010	SIZE 00026	DEN 80
PERTEC	.CM	ATTR WDSC.1	SSN 00255	SIZE 00010	DEN 63
PRINT	.CM	ATTR WDSC.1	SSN 00294	SIZE 00004	DEN 82
PROM25	.CM	ATTR WDSC.1	SSN 00323	SIZE 00006	DEN 85
RENAME	.CM	ATTR WDSC.1	SSN 00161	SIZE 00006	DEN 35
SURMIT	.CM	ATTR WDSC.1	SSN 00265	SIZE 00013	DEN 70
SYSGEN	.CM	ATTR WDSC.1	SSN 00167	SIZE 00018	DEN 40
TAPED	.CM	ATTR WDSC.1	SSN 00053	SIZE 00005	DEN 12
U	.CM	ATTR WDSC.1	SSN 00282	SIZE 00002	DEN 72
VERIFY	.CM	ATTR WDSC.1	SSN 00288	SIZE 00006	DEN 81
XREF	.CM	ATTR WDSC.1	SSN 00298	SIZE 00012	DEN 83

TOTAL NUMBER OF SECTORS: 00319
TOTAL DIRECTORY ENTRIES SHOWN: 00029

Note:

Address locations on System Diskette are subject to future revision. For update service on software changes, contact:

Microsystems Marketing
RCA Solid State
Box 3200
Somerville, N.J. 08876

Appendix M

Format of SUBMIT command

SUBMIT <filename> [<param><delim>...](CR)

BNF of command file language

<command file definition> ::= [label] <command file statement>
<line delim>...<eof>
<eof> ::=(CR)
<line delim>::=(CR)
<command file statement> ::= <MicroDOS commands>
¦ <application programs>
¦ <application program responses>
¦ <command file command>
<command file command> ::= <comment>
¦ <if command>
¦ <go command>
¦ <type command>
¦ <exit command>
¦ <read command>
¦ <Joperation command>
<comment> ::= ~C[OMMENT] <character string>
<if command> ::= ~IF <expression> <command file command>
<expression> ::= <operand> <space> <relop> <space> <operand>
<operand> ::= ~J
¦ <fparam>
¦ <numeric constant>
¦ <string constant>
<numeric constant> ::= <decimal digit> [<decimal digit>]
<decimal digit> ::= 0¦1¦2¦3¦4¦5¦6¦7¦8¦9
<string constant> ::= '<fparam>'
¦ '<character string>'
<character string> ::= [<char>...]
<char> ::= printable ASCII char including SP
<fparam> ::= ~<decimal digit>
<relop> ::= < ¦ > ¦ >= ¦ <= ¦ <> ¦ =
<go command> ::= <goto> <label>
<goto> ::= ~G[OTO]
<label> ::= %<string>
<string> ::= <anchar>[<anchar>...]
<anchar> ::= printable ASCII char excluding ~,%,$,SP
<type command> ::= ~T[YPE] <space><string>[<space><string>...]
<exit command> ::= ~E[XIT]
<read command> ::= <block read>
¦ <line read>
<block read> ::= ~K[READ]

<line read> ::= ~L[READ]
<Joperation command> ::= ~IN[CJ]
¦ ~S[ETJ] <space><numeric constant>
¦ ~D[ECJ]
<space> ::= SP