TRS-80 INFORMATION SERIES - VOLUME VI

James Lee Farvour

TRSDOS® 2.3 DECODED & OTHER MYSTERIES

Commented source and guide to **TRSDOS 2.3** for the Model I

AUTHORIZED EDITION

TRSDOS 2.3 DECODED & OTHER MYSTERIES

James Lee Farvour

TRSDOS® 2.3 DECODED & OTHER MYSTERIES

AUTHORIZED EDITION

David Moore — Editor

Charles Trapp — Assistant Editor

D. J. Smith — Cover Design and Graphics

First Edition
First Printing
November 1982
Printed in the United States of America

ISBN 0-936200-07-3

Published by
IJG Inc
1953 West
11th Street
Upland,CA
91786 (714)
946-5805

IMPORTANT

Read This Notice

Any software information is used at your own risk. Neither the PUBLISHER nor the AUTHOR assumes any responsibility or liability for loss or damages caused or alleged to be caused, directly or indirectly, by applying any information or alteration to software described in this book, including but not limited to: any interruption of service, loss of business, anticipatory profits or consequential damages resulting from the use or operation of such software information or alterations. Also, no patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the PUBLISHER and the AUTHOR assume no responsibility for errors or omissions. The reader is the sole judge of his or her skill and ability to use the information contained in this book.

Preface

This book is intended to guide the beginning or experienced system programmer through the internal operations of the TRSDOS operating system used on the Radio Shack Model I computer. A knowledge of basic computer architecture and assembly-language programming is assumed; however, the reader need not be familiar with the details of either on the Model I, as the significant features of both are presented in the text.

Chapter One describes the Model I system architecture as it pertains to the operating system implementation. A general description of the Z-80 CPU register sets and interrupt modes is included. A complete description of the Z-80 CPU and its instruction set may be found in the following manuals:

1. Radio Shack Editor/Assembler User Instruction
Manual, Catalog Number 26-2002
Available from Radio Shack

2. Z80-CPU Technical Manual
Part Number 03-0029-01
Price $7.50
Available from ZILOG Component Publications
1315 Dell Avenue
Campbell, California, 95008

Chapter One also contains a description of operating systems in general. This material is presented to give the reader a broad framework with which to compare the TRSDOS system.

All or portions of Chapter One may be skipped, depending on the experience level of the reader and the particular area of the system being studied. A detailed study of the code used for interrupt processing or any of the device drivers will require an understanding of the material contained in sections 1.1 through 1.3.

Chapter Two presents a broad overview of TRSDOS from a functional point of view. This chapter may be skipped if the reader has experience using TRSDOS.

Chapters Three through Ten discuss the individual components of TRSDOS in varying degrees of detail. Accompanying each chapter is a fully-commented source listing for the module described in the text. Following the source listings, where appropriate, are diagrams and descriptions of the data structures used by the module described.

Experienced systems programmers may read these chapters in any order, depending on their interests. Less experienced readers should begin with Chapter Three, since an understanding of nucleus-provided functions will be assumed in succeeding chapters.

Chapters Four through Ten may be read in any order.

Chapter Four describes the command line interpreter used by TRSDOS. Included in this module are subroutines used by the module described in Chapter Nine.

Chapters Five and Six describe the modules used by the file management system. The module described in Chapter Six contains subroutines referenced by the module described in Chaper Nine.

Chapter Seven discusses the error message processor used by TRSDOOS.

Chapter Eight describes the debug module available with TRSDOS.

Chapter Nine describes the module containing the bulk of the TRSDOS commands. This chapter is divided into sections by command, each command being a seperate section. These sections may be studied independently of one another. The system bootstrap loader is discussed in Chapter Ten, which may be read independently of all other chapters.

ACKNOWLEDGEMENTS

The author is especially grateful to the Tandy Corporation for allowing the source code for TRSDOS to be reproduced in this book.

Table of Contents

notes

1

1.0 Introduction

What is an operating system?

The answer to that question could fill volumes of books. This book is a partial answer to that question and a complete answer to how TRSDOS 2.3 works on the Radio Shack Model I computer. But, before plunging into TRSDOS, we need some background on the Model I system, and operating systems in general.

1.1 Model I Hardware Overview

The Model I is a Z80 based system. It comes in three configurations, the first being called LEVEL I. This configuration supports a keyboard, video, 1 cassette and memory. Memory is divided into 12K of ROM and 4K of RAM, 1K of video RAM and some special reserved addresses. This is usually referred to as a 4K LEVEL 1 system. It can be expanded to 16K of RAM and then be referred to as 16K LEVEL 1. The CPU, memory, and all control logic reside on a single board enclosed in the keyboard.

The second configuration is called LEVEL II. It has 12K of ROM and either 4K or 16K of RAM.

The third configuration is called LEVEL II with the expansion interface. The interface is used to connect peripheral units such as printers or disks to the Model I. It is a seperate enclosure containing a single board connected to the CPU by a 40-pin flat ribbon cable. The expansion interface has a floppy disk controller (WD 1771), which supports single density operations; disk addressing for four 5-1/4 inch single-sided mini disk drives; a parallel printer interface; clock interrupt logic; a screen printer interface; 2 cassette ports and sockets for 32K of memory. An RS-232 communications board which uses a breadboard area and special internal connector is also available. Maximum memory available in this configuration is 48K (plus 12K of ROM).

Standard options available from Radio Shack on all configurations are a 10-key numeric keypad and an upper/lower case option for the video. Independent

companies also manufacture options for the Model I system.

Speed up kits for the clock are available from independent sources. Depending on the kit installed, program execution times may be decreased by 30 to 50 per cent, or more. Installation of a speed up kit requires software changes to the disk operating system, or a modification to the expansion interface to physically disable the clock mod during disk operations, and sometimes during other operations.

Double-density kits are also available from Radio Shack or independent sources. These kits increase the amount of data that can be stored on a diskette from 89.9K bytes to 161.2K bytes or more. Double-density kits require the use of a non-standard operating system if double-density diskettes are to be created or read.

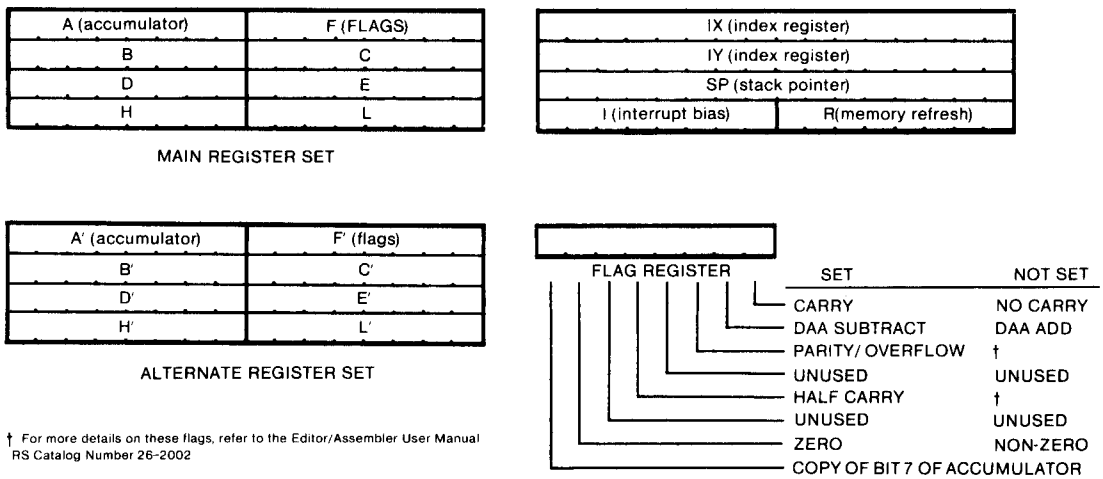
1.2 Z80 Description

The heart of the Model I system is an 8-bit Z80 microprocessor with a logical address space of 64K bytes. It is instruction compatible with the 8080A, yet it offers more instructions (158 vs. 78) and numerous other features. The CPU comes in three versions with the following clock rates: Z80 (2.5 MHz); Z80A (4.0 MHz); Z80B (6.0 MHz).

There are two sets of general purpose programmable registers. Each set consists of eight, 8-Bit registers. Either of the register sets may be selected under program control. In addition there are two, 16-Bit index registers; a single 16-Bit stack pointer register; and an 8-Bit interrupt vector bias.

Bit, byte and word (16-Bit) operations are possible between registers, or registers and memory. Following is a diagram of the registers.

Figure 1.1 *Z-80 Register Sets*



The Z80 uses a single stack pointer. It contains the address of the next available location on the stack. All stack operations use 2 bytes (16 Bits). Any instruction which saves data on the stack (e.g., PUSH, CALL, RST, . . .) causes the contents of the stack pointer to be decremented by 2 following the save operation.

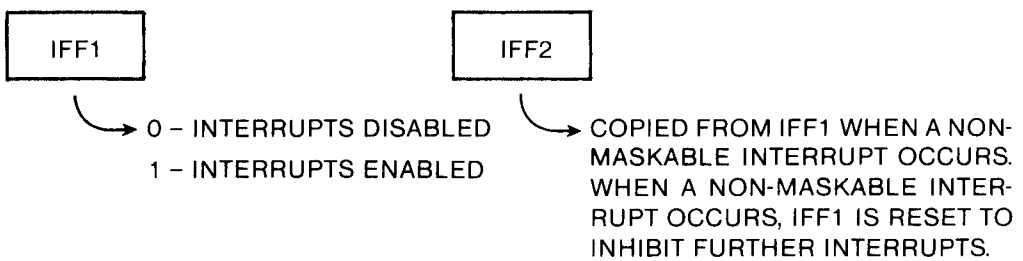
Instructions which remove data from the stack (e.g., POP, RET, . . .) cause the contents of the stack pointer to be incremented by 2 after the operation completes.

Two types of interrupts are acknowledged by the Z80. Maskable interrupts can be suppressed by use of the Disable Interrupt (DI) instruction. Non-Maskable Interrupts (NMI) cannot be supressed and are always acknowledged at the end of the current instruction cycle. Both types force the address of the next instruction onto the stack via an RST instruction.

In addition to the two general types of interrupts, the Z80 supports three modes of interrupts. Mode 0 is 8080 compatible. The interrupting device places an instruction on the data bus (usually a Restart instruction) which causes an unconditional jump to one of eight predefined addresses. Mode 1 (used on Model I systems) unconditionally forces a Restart **0038** (RST 38H) instruction onto the data bus. Mode 2 allows the interrupting peripheral to provide the lower 8 bits of an address, which are merged with the contents of the interrupt bias register to form the effective address of the interrupt service routine.

Maskable interrupts are controlled by a flip-flop labelled IFF1. A second flop labelled IFF2 is used to preserve and restore the status of IFF1 following an interrupt. The two flops are used as shown below.

Figure 1.2 *Interrupt Enable/Disable Flip Flops*



ACTION	IFF1	IFF2	REMARKS
CPU RESET	0	0	DISABLE INTERRUPTS
DI	0	0	DISABLE INTERRUPTS
EI	1	1	ENABLE INTERRUPTS
LD A,I	Unchanged	Unchanged	IFF2 TO PARITY FLAG
LD A,R	Unchanged	Unchanged	IFF2 TO PARITY FLAG
NON-MASKABLE INTERRUPT ACCEPTED	0	0	DISABLE INTERRUPTS
RETN	IFF2	Unchanged	IFF1 IS RESTORED TO ITS PREVIOUS STATUS

1.3 I/O Operations

I/O addressing in Model I systems is memory mapped except for the cassette and RS-232, which use port addressing. Video control addressing (characrer size), and cassette control logic are shared through port addressing on port **FF** hex All memory mapped addresses are listed below.

Figure 1.3 *Memory Map Addresses*

```
3801 - 38FF Keyboard Input Matrix
3C00 - 3FFF Video Display Area
37E1 - 37EF Disk Registers
37E4 - Select Cassette Drive
37E8 - Parallel Printer
37E0 - Interrupt Status Register
```

A description of the port address used on Model I systems is given below.

Figure 1.4 *Port Addresses*

```
PORT FF - Video and Cassette Control
PORT EA - RS-232 UART Status
PORT EB - RS-232 Data Latch
PORT E9 - RS-232 BAUD Rate Control
PORT E8 - RS-232 Modem Status
```

Physical I/O operations on the Model I are performed on a character basis. Direct Memory Access (DMA) is not used for any I/O operations.

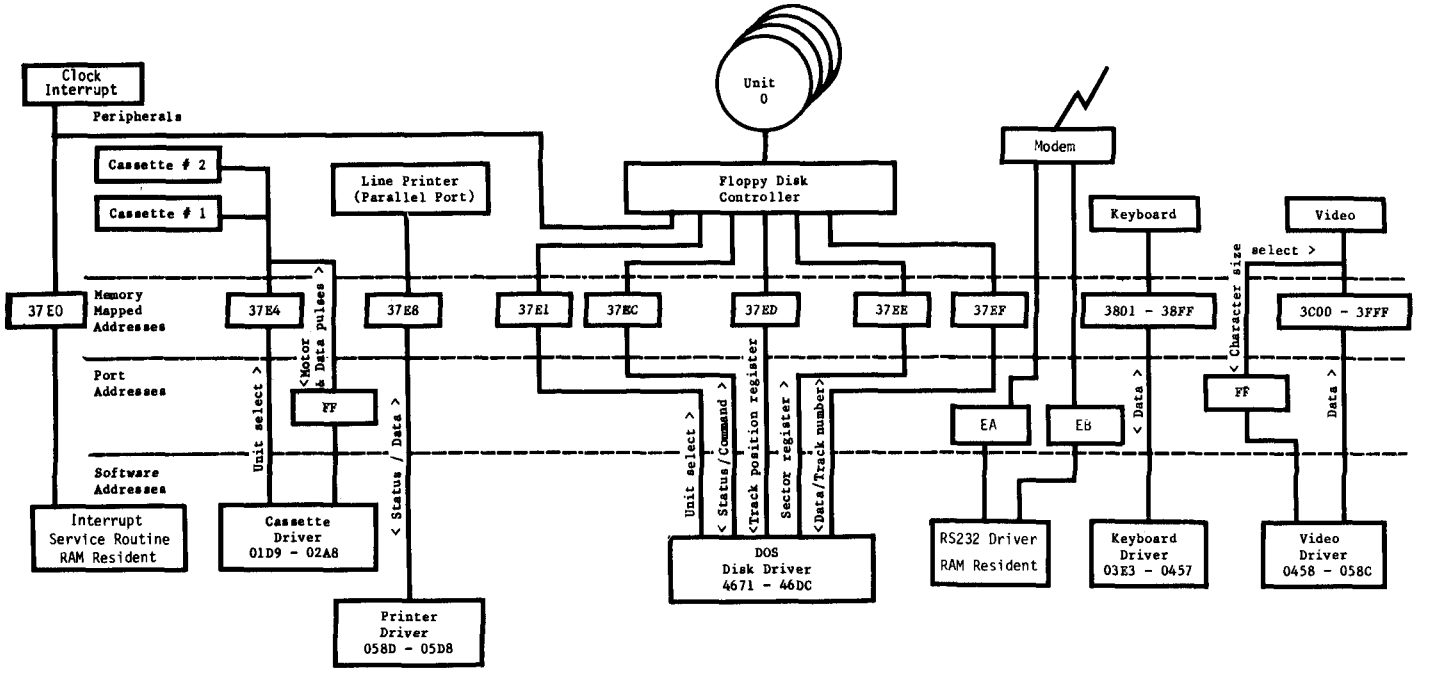
Device timing is supplied by loops in the drivers when necessary. The cassette drivers, for example, produce individual wave forms for each bit cell under software control. These waveforms are generated through OUT instruction, to port **FF**. Depending on the variable used in the OUT instruction, varying voltage levels are recorded on the cassette tape. Loops in the drivers control the duration of a particular signal level, resulting in the desired waveform.

The disk drivers also contain instructions to coordinate the timing between the CPU and the disk controller. These instructions are necessary because the command and status register for the disk controller share the same addresss. After a command has been sent to the controller (by storing into a memory mapped location), the CPU must wait at least 50 microseconds before requesting a status (by reading from the same memory mapped location), otherwise the controller may respond to the status request before acknowledging the previous command. The status would replace the command, and the change in the state of the controller would cause the last command to be lost.

Timing during disk transfers is also important because the CPU must be ready to accept or deliver each character when the controller and the disk are able to accept or deliver the next character. For this reason, and because of the software timing loops in the cassette drives, I/O operations are performed with interrupts disabled.

Memory mapped address and their drive addresses are shown below:

Figure 1.5 *Memory-mapped I/O Addresses*



1.4 Operating Systems Overview

Basically, an operating system is a program used to control the behavior of a computer. A computer is never idle, that is, the CPU is always running. So if there is nothing for the computer to do, and it is always running, then the program that is running is looking for something to do. The program that is running when there is nothing else to do is called an operating system. Sometimes they are called monitors, or executives, but the meaning is the same.

Operating systems do more than spin around in a loop waiting for the next command, but in the simplest sense that's almost all they need to do. As a bare minimum, an operating system must be able to:

1. Wait for input.
2. Compare the input to known commands.
3. Perform the required command.

We can equate the first step with the waiting for the next command loop mentioned earlier. The second step, comparing the input seems obvious, but it is

usually a distinctly separate section of code in an operating system. The third step would be represented as a series of subroutines for each of the commands supported.

There are other parts to the operating system besides the three sections listed above, but all of them would be directly related to one of those three sections. For example, 'waiting for input' implies an I/O routine for reading the keyboard. Likewise, some of the command subroutines might need to display something on the video, or read a disk. So, they too, would need special subroutines which could be shared with other parts of the system. These special routines are called 'support,' or 'ancillary,' routines.

1.5 Types Of Operating Systems

There are generally three types of operating systems: single user, multi-user and real time. Operating systems are also identified by their residency. The residency specifies where an operating system resides before it is loaded into the machine, and it usually implies something about its method of operation. Depending on the type of system, all or part of it may be loaded into memory initially. ROM systems are understood to be entirely and permenantly memory resident. LEVEL II is an example of a ROM operating system.

The process of loading and initializing the operating system is called the IPL sequence (Initial Program Load). Booting the system is another phrase commonly used for this operation. Boot loaders, or IPL programs, can be entered manually through console switches, or they may be ROM resident. ROM resident IPL's are usually executed automatically when an IPL button is pressed. IPL routines entered via switches are usually small programs that read the remainder of the IPL program from disk.

The amount of the operating system that is loaded initially will vary depending on the level of sophistication of the operating system. A simple disk resident system, for example, would be completely copied from disk to memory. The entire operating system would be memory resident.

A more advanced disk system would load a small part of the system into memory leaving the rest on disk to be loaded and executed on demand. Typically, the disk resident parts would be loaded as overlays to the resident portion. The advantage of this type of system is that it minimizes the amount of memory dedicated to the operating system, leaving more space for user programs.

There are other pieces of software, considered to be part of the operating system, which will not be discussed here because they do not co-reside with the operating system. Examples of this software are compilers, assemblers, loaders, sorts, advanced file access systems and utility programs, such as BACKUP, FORMAT, BASIC, etc They will not be discussed in any detail in this text unless they are directly related to the specific area of the operating system being discussed.

1.5.1 Single User Systems

Single user systems can support one user or terminal. This means that they can run one program at a time. The program running is either the operating system or a user's program. LEVEL II and TRSDOS are single user systems, as are most micro-computer operating systems.

1.5.2 Multi-User Systems

Multi-user systems (also known as timesharing systems) can support more than one user or terminal at the same time. That means several people can use the computer simultaneously. This type of operating system has another step in it's basic loop which was not mentioned above. That step is called schedule the next event, or job.

The normal procedure in a multi-user environment is to share the CPU between all of the users. Each user is allowed to use the CPU for a small period of time (usually measured in milliseconds). That period of time is called a time slice. The CPU is switched from one user to another at fixed intervals by the scheduler.

A scheduler is a program that controls the execution of other programs. Schedulers run at fixed intervals; they are usually driven by interrupt processing managed elsewhere in the system.

When a scheduler is run, the basic time period alloted for program execution is assumed to have expired. Other conditions can occur which cause the scheduler to be entered before a time slice has expired, but the effect is the same — the current user is to be suspended, and a new one will be selected for execution.

The selection process will be based on two criteria: the first being eligibility, and the second being priority.

Eligible jobs are those previously suspended because their time slice expired, or jobs waiting for a resource which has become available.

Within the group of eligible jobs, some method of prioritization is used to select the next job. Priority computations vary radically from system to system,, but common ingredients used in most computations include: length of time the job has been descheduled, an externally assigned priority, resources consumed and resources required.

The operating system is notified when a clock cycle expires through a mechanism called an interrupt. Interrupts cause the current program to be temporarily suspended so control can be passed to a service routine for the clock interrupt. Everything pertaining to the interrupted program is saved (on the system stack) so it can be restored when the interrupted program is restarted.

The interrupt service routine usually begins by examining a status byte containing flags indicating the cause of the interrupt. Most computers can generate more than one kind of interrupt, so a status byte is used to identify the cause of the interrupt. An alternative method for identifying interrupt causes is to vector each interrupt through a unique address, but the method used on Model I systems is to vector all interrupts through location **0038** while setting a status bit indicating the interrupt cause in location **37E0**.

In the situation described earlier, in which a clock interrupt eventually leads to a scheduling change, processing proceeds as follows. The clock interrupt would be processed initially (also called 'fielded'), by a routine designed to respond to all interrupts. It would recognize the interrupt as having come from the clock and branch to the clock service routine.

In this example, the clock service routine would count the interrupt and test if the required fixed number had occurred. If not, the interrupted program would be restarted (called returning to the point of interrupt — all registers would be restored before returning). Otherwise, control would pass to the scheduler.

The scheduler would begin by saving all register values (called the context) for the interrupted user. This is called suspending the current user. Next, it would scan a list of the suspended users and select one for execution. The register context for the next user to be scheduled would be loaded, and control of the CPU would be passed to the newly sheduled user.

Notice that control did not return to the point of interrupt after a new user had been scheduled as it did when the clock service routine determined that a scheduling call was not necessary. The reason is that if rescheduling occurs, it is assumed the interrupted program is the program to be descheduled, and hence there is no point to returning to the point of interruption until it is rescheduled.

Multi-user systems using sophisticated job schedulers are common on mini-computers and medium-to-large scale computers. They are usually called timesharing, or interactive, systems. Most micro-computer systems do not support multi-user operating systems.

1.5.3 Real Time Operating Systems

Real time systems are a form of multi-user systems. They are generally used in situations where a computer must control a number of external events (called processes) without any human intervention. Because there is usually more than one process to control, and some processes are more critical than others, a time slice system like that used in multi-user systems would not be appropriate. Furthermore, some processes are used to control others. Thus, one program must be able to schedule another. In a refinery, for example, some processes would be controlled by temperature, some by pressure and some by the completion of other process.

Programs which must run to control particular processes are usually sheduled by a master control program. The master control program is a user-written program which uses the operating system service routines to schedule and deschedule those processes as needed.

1.6 Operating System Characteristics

Operating systems may be entirely memory resident (sometimes called core resident), or partially in memory with select portions being loaded from disk when needed (called disk resident). ROM systems, for example, are nearly always core resident (all of the system is always resident in memory), while disk systems are usually partially memory resident, with the bulk of the system being disk resident. There are exceptions to both of these examples. The residency of a system is independent of its type. A real time system could be core, or disk, resident. Ditto for the single and multi-user systems.

The name of an operating system is usually derived from its residency or function. Sometimes they are combined; for example, a system will be identified as a core resident real time system.

Operating systems are also known by their specific characteristics, such as paging, mapping, and virtual memory. These particular ones are usually found on multi-user systems. They provide a means of using more physical memory than can be addressed through the 16-bit addresses commonly used on mini and micro-computers. Software features such as these usually require additional hardware to support them.

Other software features associated with operating systems that do not require hardware would be swapping, file overflow, and file and record management facilities. Features which can be implemented with or without extra hardwre are multi-mainframe, and shared-file access.

Another significant area of operating system software is computer communications. There are two types of computer commuunications. The first is terminal to computer communications. Terminals are input devices connected to the computer (usuall called a host system) by telephone lines or through a direct connection using a current loop or RS-232 hook up. Other types of connections are also used. The terminal can be anything from a CRT and keyboard, such as those used on time sharing systems, to a remote job entry (RJE) station consisting of a card reader, operators console and line printer.

The second type of communication involves computer to computer communication. This type of communication is usually distinguished from the first type for several reasons. First, the rate of transmission is usually much higher. Terminal or RJE communications rarely involve transmission rates higher than 9600 BAUD (bits per second), with the usual range somewhere between 300 and 9600 BAUD.

Second, the physical method used for the transmission of data between computers is normally different from that used for the lower-speed communication.

In low-speed communications, each character is preceded by a start bit signal, and followed by a stop bit signal. These signal bits serve as timing marks for the transmission equiptment. In high-speed communications, data is transmitted as a block of characters with a synchronizing clock pulse.

A third significant difference in the manner in which the data is transmitted on low-speed systems is that transmission is on a character-by-character basis. High-speed systems transmit data in blocks containing many characters.

notes

2

2.0 TRSDOS Overview

Model I systems with disks have two operating systems. The first is a core-resident, single-user system called LEVEL II.

LEVEL II is a ROM-based system occupying the first 12K of memory (locations **0000 - 2FFF** hex.). It uses RAM from **4000 - 4200** as a scratch pad area.

Primarily, LEVEL II is a BASIC interpreter with a few built-in system commands, such as an editor and a cassette dump and load utility. Embedded in Level II are drivers for a parallel printer, the keyboard, the video display and the tape cassette. It also contains system initialization code which is executed when the system is initially turned on or restarted.

LEVEL II is entered automatically at address **0000** when POWER ON is activated, or at address **0066** when RESET is activated. Both entry points branch to a system initialization sequence which determines if a disk controller is present. If there is no disk controller, the LEVEL II system is initialized and the BASIC interpreter is entered. If a disk controller is detected, a bootstrap program is read from the disk and executed. This bootstrap will load and initialize the disk operating system called TRSDOS.

The second operating system used on Model I systems is TRSDOS.

TRSDOS is a single-user, disk-resident operating system. It is composed of a core-resident nucleus and disk-resident system overlays and utilities. The minimum configuration required for TRSDOS is 32K bytes of memory, and one disk drive which must be addressable as drive zero. In order to properly run, TRSDOS requires a system diskette to be mounted in drive zero at all times. TRSDOS supports any memory size from 32K to 48K bytes of memory and up to four, 35-track, single-sided, single-density, 5-1/4 floppy disk drives.

As stated earlier, TRSDOS is loaded by a bootstrap program which is read from disk by LEVEL II. The portion of TRSDOS initially loaded is called the nucleus.

The nucleus occupies memory from **4200 - 4E00**. Portions of it are also loaded into the **4000 - 4200** region, and the system initialization code is loaded at **4E00**. The code loaded into the **4000 - 4200** region consists of jump vectors used for calling various nucleus functions, data structures and storage areas used by the nucleus and the remainder of the system. The system initialization code loaded at **4E00** is executed immediately after the system has been loaded.

The bulk of TRSDOS resides on the system disk. The disk resident portion consists of overlay modules and system utility programs.

Overlay modules are loaded on a demand basis by the nucleus. They contain code for system functions which need not be permanently core-resident. Examples of these functions are: the command line processor; OPEN and CLOSE routines plus select system commands, such as DEBUG, TRACE and BASIC 2.

The overlay area begins at the end of the nucleus (**4E00**) and ends at the start of the utility load area (**5200**). Only one overlay is loaded at a time. Overlays could be constructed which call secondary overlays; however, this is not a standard practice in TRSDOS.

System utility programs such as BACKUP and FORMAT are loaded at address **5200**. SYS6, which is technically an overlay module, is also loaded at **5200**. It contains code for the DOS commands AUTO, KILL, DATE and TIME, to name a few. Utility programs may make nucleus calls which result in an overlay being loaded; thus, they must reside in separate areas of memory.

The system utility programs and overlays exist as files on the system diskette. Since they are assigned the invisible attributes, "SEE TRSDOS MANUAL," they are not displayed when the DIR command is used unless the 'I' (invisible) parameter is specified.

Overlay files have a SYS suffix, while utility program files have a CMD suffix. The exception is SYS6/SYS, which is loaded into the utility area.

SYS6 is also the only module in the system with multiple entry points. The editor/assembler and loader used by TRSDOS do not support relocatable code, multiple entry points or symbolic external references. SYS6 does not have multiple entry points in the traditional sense of the word, but it achieves the same effect by being called with a command index value, which specifies which of its nineteen commands is to be executed.

A command index is passed to SYS6 in the C register by the program that requested it to be loaded. In the usual case, the command line interpreter (SYS1/SYS) is the caller.

Utility programs fetch their parameters from the command line used to call the utility. Upon entry, the HL register points to the first character after the command specification in the command line buffer.

TRSDOS is not an interrupt driven system; however, it does support interrupts. A clock interrupt is generated every 25 milliseconds on systems with an expansion interface. This interrupt, plus any spurious interrupts, are fielded by a general purpose service routine in TRSDOS.

Upon detection of a clock interrupt, the clock interrupt processor is entered. This processor executes sub-tasks indicated by an interrupt task list. Initially, the task list is empty. System commands for the CLOCK and TRACE functions require use of two of twelve entries in this list.

The remainder of the entries in the list are unused. Subroutines in the nucleus are defined, which will add or delete entries to this list. This means some sort of scheduling or multi-tasking function could easily be added as an application program.

The nucleus contains other useful subroutines which can be called from users programs. They will not be detailed here, but a complete list of nucleus entry points can be found in Section 3.4.14.

The major features of TRSDOS are its file management system, the disk space manager, the file loader, the overlay loader, interrupt service routines and the various command processors. Each of those features is discussed in detail in subsequent sections. The following descriptions of some of these features is very general and is intended to acquaint the reader with the external characteristics of these features.

2.1 File Management System

The file management system provides a mechanism for accessing files by name. Each diskette has a directory on track 17 (dec.) which is used to keep a record of file names, their disk addresses and passwords. Whenever a file is referenced, the directory is searched until the named file is located or all directory entries have been examined. Specific commands related to the file management system are:

```
ATTRIB - assign file attributes
DIR    - list directory contents
KILL   - delete a file from the directory
FREE   - return amount of available disk space
PROT   - assign diskette password to all files
RENAME - change file name
```

2.2 Disk I/O Services

Disk services in TRSDOS support the file management system. They create and delete directory entries, allocate disk space, translate relative record numbers into absolute track and sector addresses, rewind files, position files to specified records, test for end of file conditions, and call the disk driver to perform data transfers. The disk management system does not support: file overflow from one device to another, any sector size other than 256 (dec.) bytes, or the ability to bypass irrecoverable errors.

Embedded in the disk services is an elementary record manager. It provides record movement between the user and system data areas, as well as packing and unpacking services.

Two record types are supported. They are: fixed-length physical records (always one sector), and logical records. Logical records can vary in size from 1 to 255 bytes in length. All logical records in a file must be the same length except for DISK BASIC files. DISK BASIC supports variable length logical records because it has

its own record management routines.

Disk space is allocated on an as-needed basis in units called granules where 1 granule equals 5 consecutive sectors. A permanent record of the assigned and available granules is maintained in a special sector of the disk directory. Each file entry in the directory has a list of assigned granules. When a file is purged (KILL'ed in TRSDOS nomenclature), the granules assigned to the file are returned to the list of available granules.

2.3 TRSDOS Utilities

Utilities are special purpose programs, supplied with an operating system, which are not an integral part of the operating system. They are seperate binary modules, loaded into an area of memory not used by the system. Utility programs on TRSDOS include the following:

```
BACKUP   - disketted copy program
FORMAT   - formats diskettes
BASIC    - DISK BASIC extension for LEVEL II
DISKDUMP - disk sector dump program
TAPEDISK - copies cassette to disk
```

BASIC and DISKDUMP do not conform to the definition of a utility as given above, but they are utility type programs. BASIC is a assembly language program which is an extension to LEVEL II BASIC. After being loaded, BASIC modifies pointers in the **4000 - 4200** region to indicate that DISK BASIC has been loaded before transferring control to LEVEL II BASIC. If an exit from LEVEL II to TRSDOS is taken, and another utility such as BACKUP is loaded, the BASIC utility loaded earlier would be destroyed.

DISKDUMP, is a utility program written in DISK BASIC. The BASIC utility program would need to be run before this utility could be run.

2.4 System Commands

System commands are similar to utility programs. The difference is that they are recognized and acted upon directly by the system nucleus. The code for system commands is always resident in the system (either in the nucleus or in one of the overlay modules on disk). TRSDOS system commands include the following (plus all of the file management commands):

```
AUTO   - define startup procedure
CLOCK  - display current time
COPY   - file copy request
DATE   - enter date
DEVICE - list device assignments
LIB    - list all system commands
LOAD   - load object file
LIST   - list text file on video
PRINT  - print text file on video
TIME   - enter time for CLOCK display
TRACE  - display P register
VERIFY - force verify on all writes
```

2.5 Directory Data Structures

TRSDOS recognizes two types of diskettes — system diskettes and data diskettes. System diskettes have a boot loader on track 0, sector 0, plus a directory entry for the nucleus file (SYS0/SYS). Data diskettes have a boot loader on track 0, sector 0, but lack the nucleus file. Both diskettes have the same directory structure. The directory occuppies one track (17 dec.). It has two overhead sectors, leaving eight sectors for file entrys. Each directory sector can hold up to 8 file entries.

The overhead sectors are the HIT (Hash Index Table, sector 1), and the GAT (Granule Allocation Table, sector 0). The HIT contains a hash code for every file on a diskette. When a reference is made to a file, its hash code is computed and matched against those in the HIT. The position of the matching hash code in the HIT gives the starting sector number for searching the directory. This speeds the process for locating a file considerably, since the directory is not searched unless the file exists, and the search begins at an advanced point in the directory.

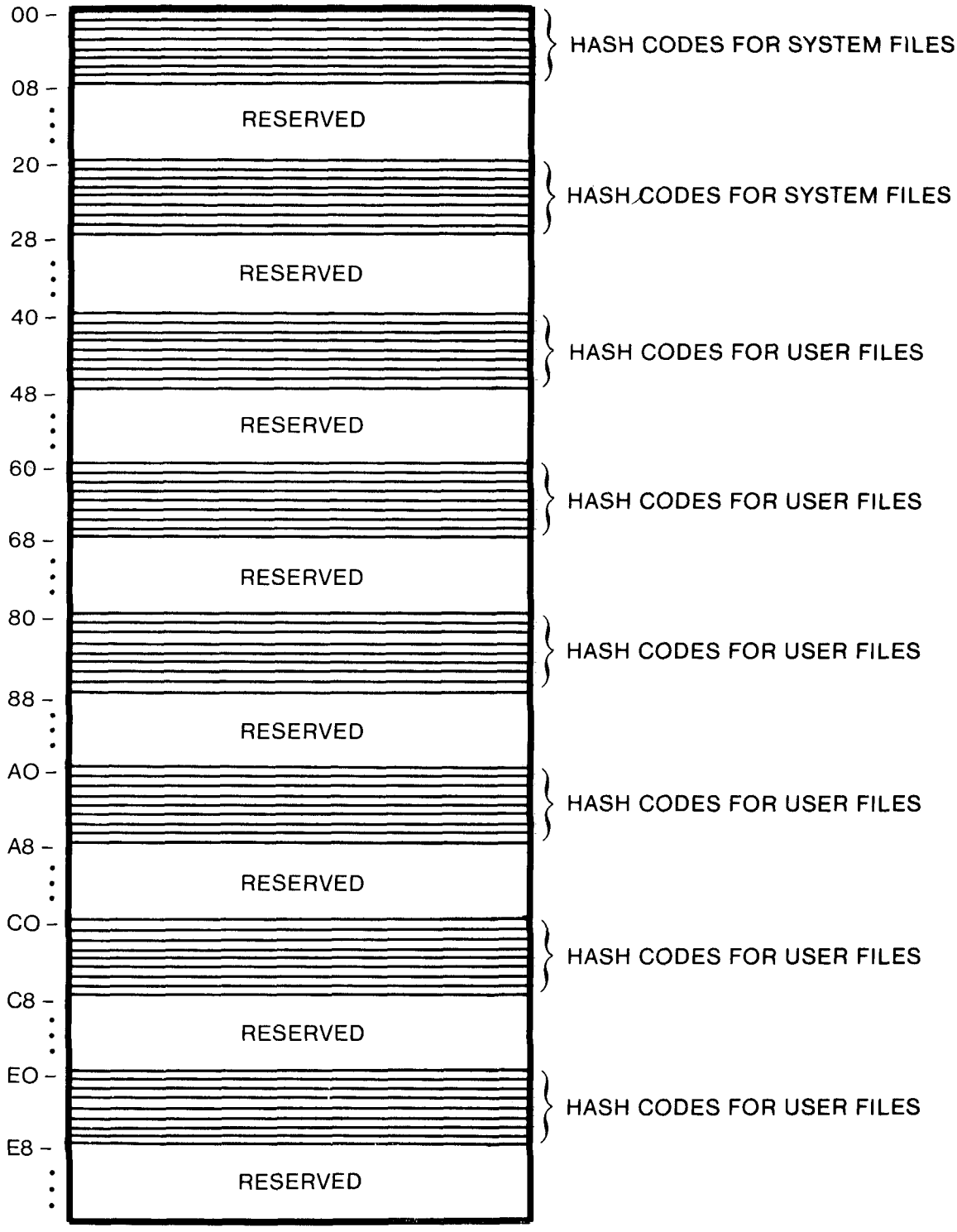
Hash codes are computed according to the following code:

Figure 2.1 *Hash Code Computation*

```
        LD      B,11        NO. OF CHARS TO HASH
        LD      C,0         ZERO HASH REGISTER
LOOP    LD      A,(DE)      GET ONE CHAR OF NAME
        INC     DE          BUMP TO NEXT CHAR
        XOR     C           HASH REG. XOR. NEXT CHAR
        RLCA                2*(HR. XOR. NC)
        LD      C,A         NEW HR
        DJNZ    LOOP        HASH ALL CHARS
        LD      A,C         GET HASH VALUE
        OR      A           DON'T ALLOW ZERO
        JMP     DONE        EXIT, HASH IN A
        INC     A           FORCE HASH TO 1
DONE      .                 EXIT, HASH IN A
          .
          .
```

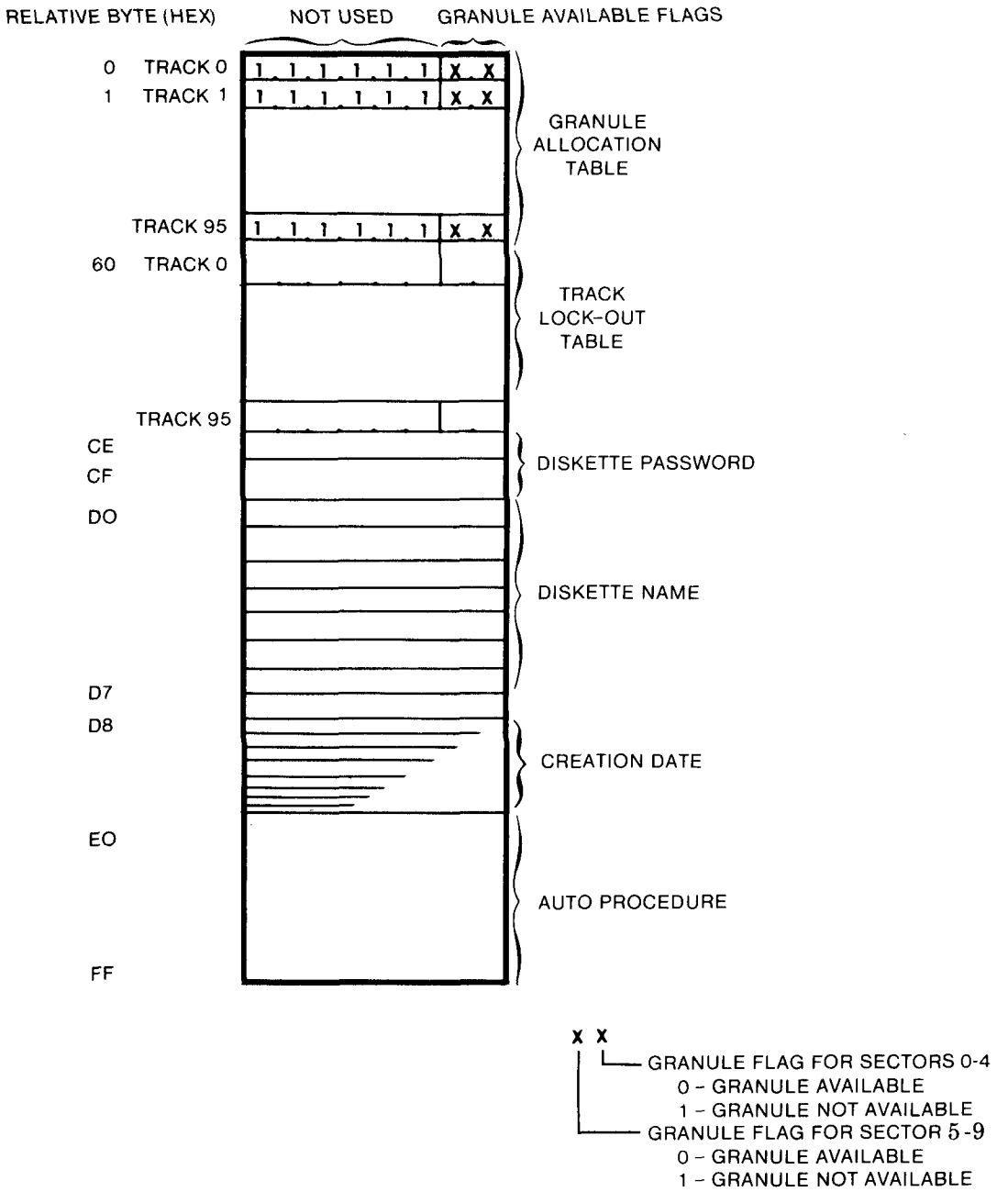
The format of the HIT is shown below:

Figure 2.2 *Hit Sector Description, Track 11 Sector1*



The second overhead sector in the directory is the GAT. This sector contains a bit map indicating which areas of the disk are available, the creation date, a master diskette password, and an optional command to be executed at system start up (AUTO feature). A description of the GAT sector is given below.

Figure 2.3 *GAT Sector Description, Track 11, Sector 0*



Sectors two through nine contain the file entrys. Each sector contains eight 32-byte entries. There are two types of directory entries: primary and overflow. The majority of directory entries are primary entries. These contain the file name, passwords, file attributes, and granule allocation pairs.

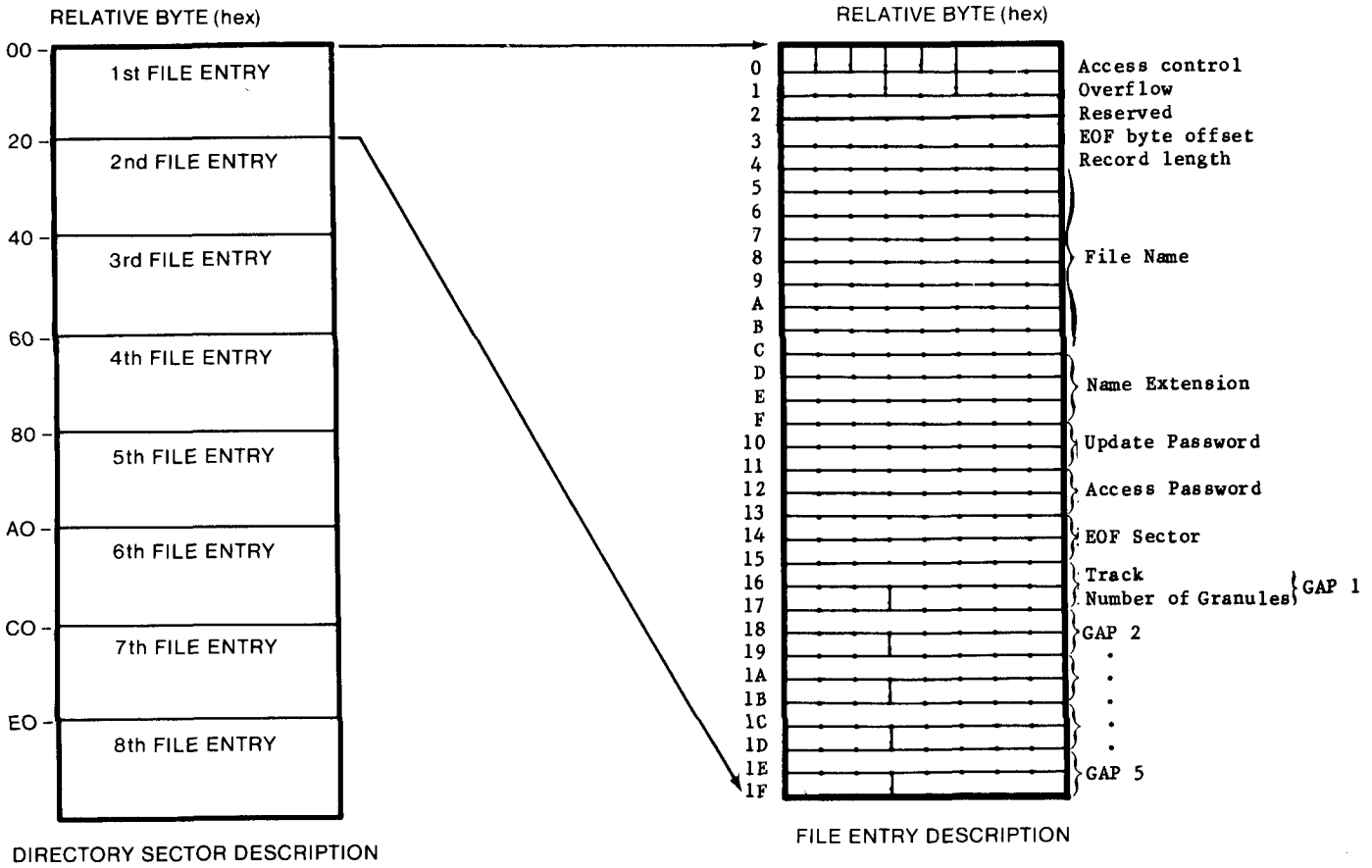
Overflow entries are created when there are more granule alloction pairs assigned to a file than will fit in the primary directory entry.

The first two entries in every directory sector (2 to 9) are reserved for system use. Thus, the maximum number of user files which can be saved on a diskette is 48.

48 = Number of directory sectors (8) x number of entries available in each sector (6).

The format of the directory sectors is given below:

Figure 2.4 *Directory Sector/File Entry Description*



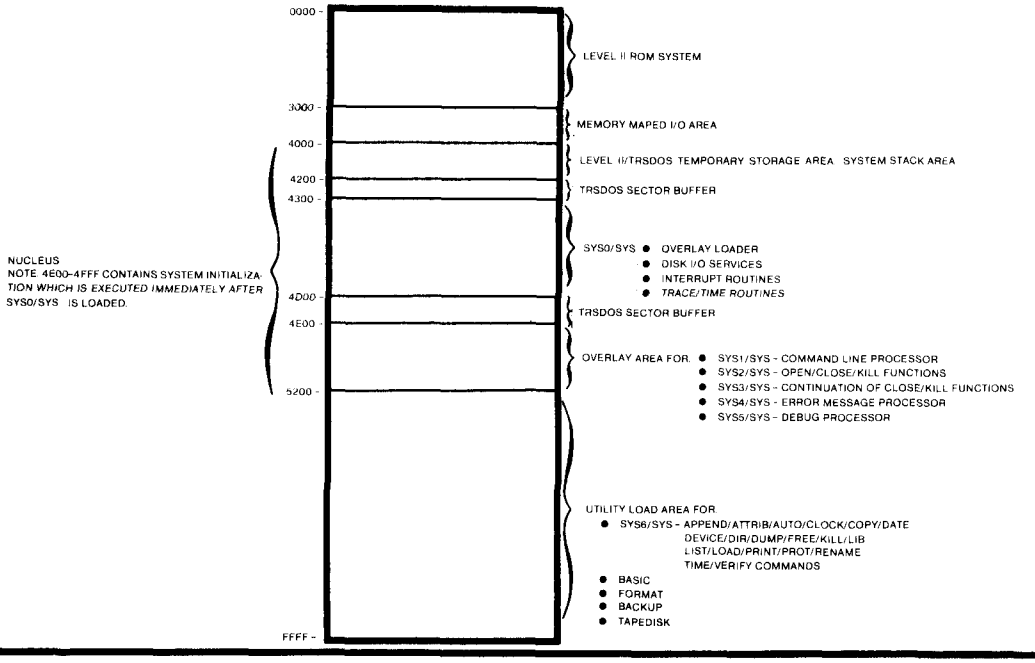
3

3.0 SYS0/SYS

TRSDOS is a disk resident, overlaid system. A small part of it (called the nucleus), is core resident, while the remainder resides on disk. The nucleus contains: an overlay loader, disk I/O services, the disk driver, TRACE and TIME displays, the interrupt service routine, all core resident data structures, 2 sector buffers and system initialization code.

All system command processors and utility programs are disk resident. The command processors are loaded as overlays above the nucleus (but below the utility load area). Utilities are loaded at **5200**, which is considered the first available area of memory for user programs.

Figure 3.1 *Nucleus Description*



All system modules, including the nucleus and the overlays, exist on the disk as named files. The nucleus is named SYS0/SYS; The command overlays are named SYS1/SYS,. . . SYS6/SYS.

3.1 Loading the Nucleus

When the system is brought up or RESET is activated, control passes to LEVEL II. LEVEL II immediately determines if the system has disks. If it does, a special program called a boot loader is read from disk. The boot loader is used to load the nucleus for the disk operating system. Following is the code used by LEVEL II to load the boot loader:

Figure 3.2 *Code From 0696 to 06C9 in LEVEL II*

```
0696 3AEC37   LD      A,(37ECH)   --- Load disk status
0699 3C       INC     A           --- Test for Expansion Interface
069A FE02     CP      02H         --- and disk drive
069C DA7500   JP      C,0075H     --- Go if no disk
069F 3E01     LD      A,01H       --- Unit select mask for drive 0
06A1 32E137   LD      (37E1H),A   --- Select drive 0
06A4 21EC37   LD      HL,37ECH    --- Addr of disk command / status register
06A7 11EF37   LD      DE,37EFH    --- Addr of disk data register
06AA 3603     LD      (HL),03H    --- 3 to disk command register = restore, position
06AC 010000   LD      BC,0000H    --- Delay count                          :to track 0
06AF CD6000   CALL    0060H       --- Delay for approx 3 seconds
06B2 CB46     BIT     00H,(HL)    --- Test if controller busy,
06B4 20FC     JR      NZ,06B2H    --- Loop till not busy
06B6 AF       XOR     A           --- 0 to A
06B7 32EE37   LD      (37EEH),A   --- 0 to sector register
06BA 010042   LD      BC,4200H    --- BC = addr of buffer area
06BD 3E8C     LD      A,8CH       --- A = read command
06BF 77       LD      (HL),A      --- Read sector 0, track 0 into 4200 - 4455
06C0 CB4E     BIT     01H,(HL)    --- Test if data ready
06C2 28FC     JR      Z,06C0H     --- Go if no data avail
06C4 1A       LD      A,(DE)      --- Get next byte from disk
06C5 02       LD      (BC),A      --- Transfer data to 4200+
06C6 0C       INC     C           --- Bump buffer pntr
06C7 20F7     JR      NZ,06C0H    --- Go if not 256 bytes
06C9 C30042   JP      4200H       --- Done, transfer to TRSDOS loader
```

There are several aspects of the boot program that make it unique. Like the rest of the system programs, it exists as a named file. But because the space in LEVEL II used for reading the boot is restriced, it is assumed to exist at track 0, sector 0. Furthermore, it is an absolute core image program, one sector long. Thus it can be read directly into memory and executed.

The boot is the only core image program in TRSDOS. Binary modules produced by the editor assembler have embedded control codes indicating the load address, length of string to be loaded and a execution address. The core image of the boot was created by writing a section of memory containing the boot to disk.

Control is passed to the boot after it has been read into locations **4200 - 42FF**. Notice that this is one of the sector buffer address for the nucleus, so there is no conflict between the addresses for the boot and those of the nucleus. The boot then finds the disk address of SYS0/SYS (the nucleus), and loads it into memory. Since SYS0/SYS is not a core image program, the boot must be able to interpret the loader codes, and it must contain its own disk I/O routines, as well as a sector buffer. The sector buffer address is **4D00 - 4DFF**; again there is no conflict with code in the nucleus because it has a sector buffer at the same address.

The boot has a limited facility for interpreting the granule allocation pairs in the directory describing the disk addresses for the nucleus. Therefore, it cannot have more than one granule allocation pair. For a detailed description of the boot loader, see Section 10.

3.2 System Initialization

System initialization is performed immediately after the nucleus has been loaded. Operations performed during this stage consist of the following: initializing the **4300** region, setting up the system stack pointer, determining memory size, initializing the device driver tables, and executing any procedure file created with the AUTO command.

Since this code is executed only once, the space it occupies can be used for other purposes after the initialization process has completed. In this case, the code begins at **4E00**, which is also the start of the nucleus overlay area. Because this code resides in the overlay area, it would be impossible to execute a nucleus overlay as part of the stytem start up initialization.

Beginning at **4E00**, interrupts are disabled, and interrupt mode 1 is selected. Selection of interrupt mode 1 means that when interrupts are enabled, all interrupts will be vectored through location **0038**.

Next, the utility load address is initialized to **5200**, and the memory size is computed by the loop at **4E0F - 4E1B**. When the highest RAM adddress is found, it is saved in **4049**.

The code from **4E20 - 4E55** initializes the device address table (**43B8 - 43BF**) and the device mnemonic table (**43C0 - 43CC**). These tables are used by the DEVICE command to report the device configuration and the driver addresses for all devices listed in the table.

Starting at **4E20**, the address of the replacement keyboard driver (**43D8 - 43FB**) replaces the address of the ROM driver in the keyboard DCB, as well as being stored in both device tables. Next, the keyboard RAM buffer address is saved in the device driver table.

Continuing at **4E2F**, the video driver address and the video RAM buffer are both copied to the device tables. Next, the printer driver address and its buffer address are copied to the device tables. Finally, the list of driver and buffer addresses is terminated with a zero byte.

Beginning at **4E57**, the keyboard DCB is copied to **4358 - 435F**, then a loop is executed which zeros the 8 bytes after the DCB. A second loop starting at **4E6A** stores 16 **FF**'s after the 8 bytes of zeros. The reason for this initialization is not known.

There is some unused code in the section just described. The INC, JR, and LD instructions at **4E52 - 4E55**, as well as the PUSH at **4E52** and the POP, DEC, and JR at **4E72 - 4E75** serve no useful purpose. For that matter, neither does the LD instruction which begins the next section to be described.

Starting at **4E79**, **4015** (the first byte of the keyboard DCB) is set to zero, and the interrupt mask control byte is zeroed. Zeroing the interrupt mask byte will cause any interrupts which might slip in (very unlikely since interrupts are inhibited) to be discarded.

Continuing at **4E81**, a JP 4518 instruction is synthesized and stored into **4012**. This replaces a DI/RET sequence placed there by the LEVEL II ROM code prior to the loading of the nucleus. This causes control to pass to **4518** when an interrupt is recieved.

Next, the address of the clock interrupt routine (**4560**), and the disk interrupt routine **45F7** are stored in the interrupt scan list at **405B** and **4059**, respectively.

Following that, the interrupt mask control byte at **404C** is updated to indicate interrupt routines exist for both clock and disk interrupts.

At **4E9F**, the DEBUG control byte is initialized to a positive value (DEBUG not active), and **45AF** (address of the clock maintenance code) is stored in the interrupt task list by calling a system service routine at **4410**. The maintenance routine will be executed evey 200 milliseconds under the control of the interrupt processor. It updates the binary clock values maintained in **4041 - 4043**; however, the clock value will only be displayed if the CLOCK ON command is executed.

Upon return, the message 'TRSDOS 2.3 . . . .' is displayed by the loop from **4EAF - 4EB8**. This code is followed by a call to scan the keyboard. If the carriage return key is active, the AUTO procedure test is skipped, and control passes immediately to **4400**, where a call to load and execute SYS1 is performed.

If the carriage return key is not active, the code at **4EC2 - 4ECB** is executed to read sector 1 of track 11 into the buffer at **4200**. If an error occurs during the read, control goes to **4409**, where SYS4 is called to display an error message

Assuming no read error, the buffer is examined to determine if an AUTO procedure has been defined. A carriage return at **42E0** (the **E0**'th byte of the sector) means no AUTO procedure file exists. In this case also, control passes to **4400**, which causes SYS1 (the command line processor) to be loaded and executed.

If **42E0** does not contain a carriage return, it is assumed to hold a 20-character command to be executed. This command is copied to the commmand line buffer by the code at **4ED9** - **4EE2**, displayed on the video, and then SYS1 is called to process the command.

When the initialization code completes, it exits by jumping to **4400** or **4405** to load and execute SYS1. At that time, SYS1 will either wait for the next command or process the one in the command line buffer.

3.3 Command Line Processing

At the end of system initialization, the command line intrepreter overlay (SYS1/SYS) is loaded. This overlay displays the message:

DOS READY

then posts a read (waits for next input) against the keyboard waiting for the first command. When a command is entered (carriage return struck) SYS1/SYS compares the command against a list of system commands. If a match is found, the overlay for the command is loaded and executed. When the command overlay completes, SYS1/SYS is reloaded to read and process the next command.

If a command line does not match one of the system commands, it could be the name of a utility program, such as BACKUP, FORMAT, or a user-written assembly language program. SYS1 determines if this is the case by searching the directory for a file name matching the command line. If one is found, the file is loaded and control is passed to the transfer address. When processing completes, the utility (or user program) must return to **402D** so that SYS1 can be reloaded.

If no match is found between the command and the file names in the directory, the message:

PROGRAM NOT FOUND

is displayed, and another keyboard read is posted.

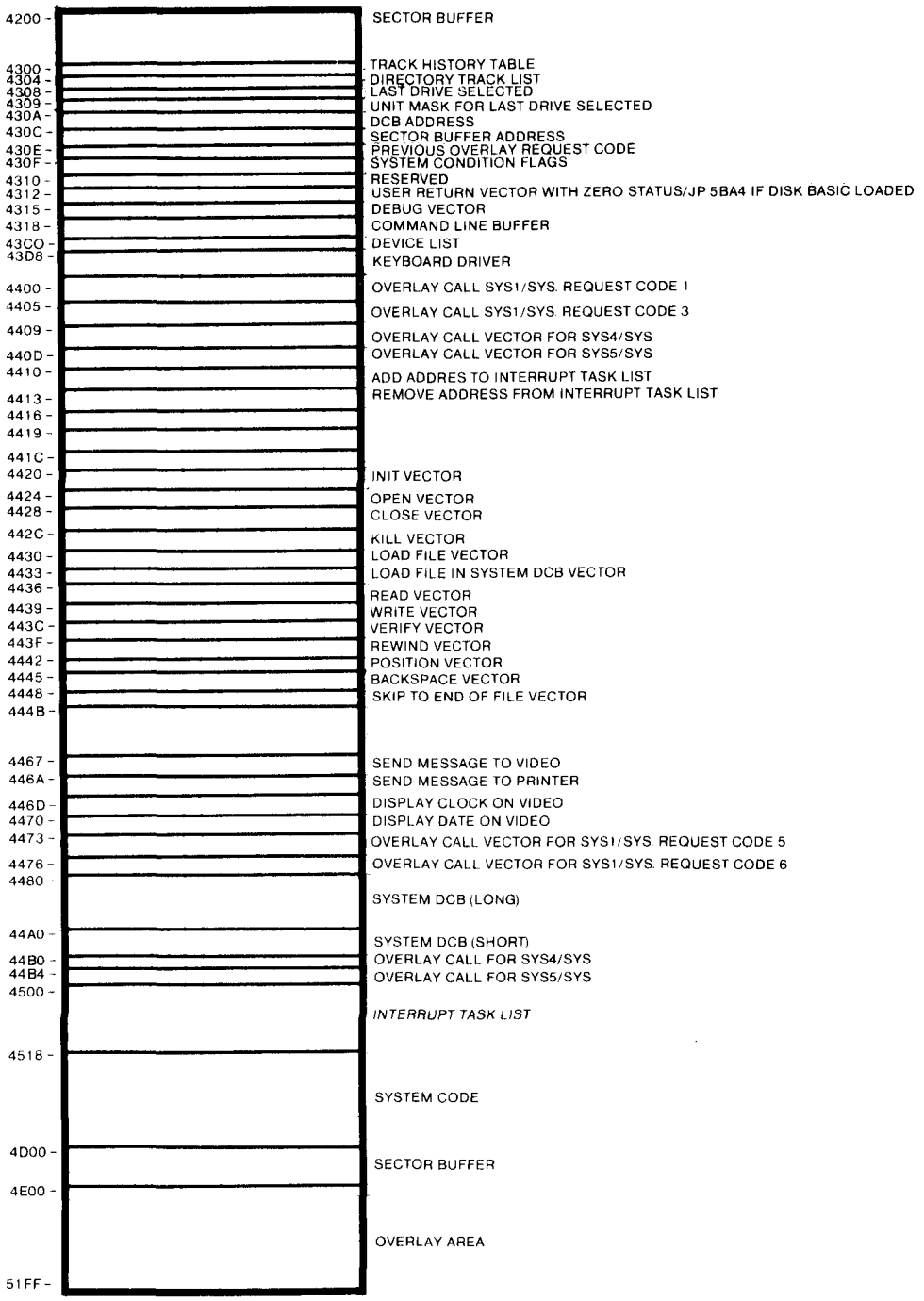
System overlays also contain 'subroutines,' which may be invoked indirectly through system calls to OPEN, CLOSE, or KILL, to name a few. When one of these calls is made (from a utility or user program to one of the fixed addresses in the nucleus), SYS0 loads the appropriate overlay and transfers control to it. Since the overlays are considered an extension to SYS0, they know of (and make use of) many subroutines in the nucleus. Thus, SYS0 and the overlays can be thought of as one program. For a detailed description of SYS1/SYS see Section 4.0.

3.4 Nucleus Functions

The nucleus is the core resident portion of TRSDOS. It contains code for the following functions: interrupt processing, disk services, a file loader, trace display,

clock display and the keyboard driver. In addition, data structures used throughout the system are maintained in the nucleus. Those structures are: a jump table for nucleus funtions, the command line buffer, a disk track history table and interrupt processor addresses. The memory map below lists the major entry points in the nucleus.

Figure 3.3 *Nucleus Entry Points*



3.4.1 Interrupt Processing

Interrupts notify the computer that an external event has taken place. This causes the current sequence of instructions to be suspended so that control may be passed to a routine which can process the interrupt. Usually, this action is the completion of an I/O request, or the passage of a fixed interval of time. The former are called I/O interrupts, while the latter are referred to as clock interrupts.

There are two types of interrupts: maskable and non-maskable. Maskable interrupts can be suppressed through the use of the disable interrupt instruction. In order for maskable interrupts to be acknowledged, the CPU must be enabled to accept interrupts.

Maskable interrupts, on a Model I system cannot be stacked. For example, if the CPU is disabled for interrupt processing and five clock interrupts occur, only the first will be remembered. When interrupts are re-enabled, one clock interrupt will be generated rather than five; the others would be lost.

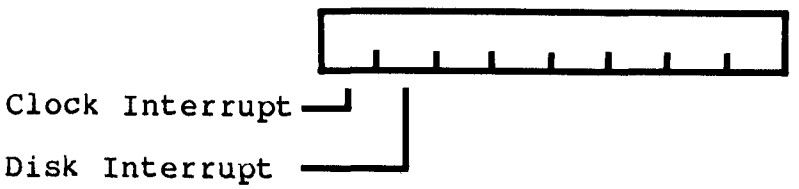
Non-maskable interrupts cannot be suppressed. Regardless of the state of the interrupt flip-flop, they will be recognized. Non-maskable interrupts on the Model I system are generated when the POWER ON or RESET buttons are activated.

When an interrupt is acknowledged, three things happen. First, the P register is saved on the stack. This is done automatically when the interrupt cycle is started. The P register is saved so that after interrupt processing has completed, control can be returned to the point where the interrupt occurred.

Second, a new address is substituted for the current P register. The address will vary depending on the type of interrupt being acknowledged (maskable, or non-maskable) and the interrupt mode selected (the Z-80 supports three modes, see Section 1.2). These addresses are usually called interrupt vector addresses. On Model I systems, maskable interrupts force the P register to **0038**.

Third, if the interrupt is maskable, a bit is set in location **37E0**, indicating the exact cause of the interrupts. The format of the interrupt status byte is shown below:

Figure 3.4 *Interrupt Status Byte*



Non-maskable interrupts generate a vector address of **0000** or **0066** for a POWER ON or RESET condition, respectively. Maskable interrupts cause an RST 38 instruction to be executed. In all cases the vector addresses are defined in LEVEL II ROM. A list of interrupt addresses follows:

Figure 3.5 *Interrupt Vectors*

ACTION	VECTOR ADDRESS	PROCESSING
POWER ON	0000	Initial system start up.
RESET	0066	System start up for disk systems. System restart for ROM based systems.
CLOCK/DISK	0038	See below.

The code executed instantaneously in reaction to an interrupt is called the interrupt processor. The interrupt processor determines the exact cause for the interrupt and passes control to an interrupt service routine designed for each specific condition. The service routines may call other routines that need to be executed on a periodic or exception basis. In a time-sharing operating system, for example, the clock service routine might call the job scheduler every tenth time, rather than on every clock interrupt.

Figure 3.6 *Interrupt Vectors*

NO INTERFACE	INTERFACE PRESENT
DI RET	JP 4518 -

Maskable interrupts on the Model I are generated by either the clock or disk. Disk interrupts are not used, that is, they are not explicity requested by the system except in a few special cases to clear the controller. Therefore, there is no special handling for disk interrupts; they are acknowledged and control is returned to the point of interrupt.

Clock interrupts are generated automatically every 25 milliseconds. TRSDOS uses these interrupts to drive internal functions, such as maintaining a time of day clock and displaying the P register for the TRACE command.

Interrupt processing takes place on two levels. The first level accecpts the interrupt, saves the current register context, determines the exact cause of the interrupt and passes control to the second level.

The second level consists of routines designed to process a specific type of interrupt. TRSDOS has second level routines for clock and disk interrupts.

Second level routines may call a third level, which in turn could call a fourth level, and so on. The disk interrupt routine, for example, does not call any other levels, but the clock interrupt calls a third level which will eventually call a fourth level. See Sections 3.4.2 and 3.4.3 for a description of the second level disk and clock processing routines. The remainder of this section discusses first level processing.

The interrupt processor for maskable interrupts begins at location **4518**. When entered, the return address has been pushed onto the stack, and interrupts are disabled. It begins by saving the HL and AF registers on the stack, and the interrupt status byte (**37E0**) in **404B**. The status must be saved because reading it forces it to zero.

Next, the status is compared to the interrupt mask at **404C** on a bit-by-bit basis. This mask indicates which interrupts can be serviced by having a 'one' in the position corresponding to the interrupt status bit. The initial value for this byte is **C0**, which indicates clock and disk interrupts are supported. If there is no match between the status byte and the interrupt mask, the interrupt is discarded and control returned to the point of interrupt. Before returning, the BREAK key is tested, if it is inactive the registers are restored, interrupts are enabled, and control returns to the point of interrupt.

When an active BREAK key is found, a test is made to determine if the DEBUG utility has been called. If it has, DEBUG is loaded and executed. This test is accomplished by examining the contents of **4015** (if it is zero, DEBUG has not been called; otherwise, it has). If the DEBUG flag is non-zero, interrupts are enabled, and control is passed to the address contained in locations **4316 - 4317**. This is usually the address of a location in the nucleus that makes a loader request for DEBUG. This is how DEBUG gets loaded when BREAK is pressed and DEBUG is not already memory resident.

Note that the context saved when the interrupt processer was entered is restored before DEBUG is entered; however, the interrupt return address is left on the stack. This means that control can be returned to the point of interrupt. This is a significant debugging feature, since it means an executing program can be interrupted and restarted at the point of interrupt. On the other hand, if DEBUG were entered capriciously with control being returned to **402D**, the system stack will grow because of the context pushes and eventually overwrite critical areas of low memory.

If a match is found between the interrupt status and any of the bits in the mask byte, then a system subroutine to service the interrupt does exist. The interrupt processor then determines the exact cause of the interrupt and passes control to its service routine. When the service routine completes, control is returned to the interrupt processor so the rest of the bits in the status byte can be tested (it is possible for several interrupts to occur simultaneously). A list of the addresses of the service routines is kept in loacations **404D - 405C.**

This is done by testing the status word on a bit-by-bit basis starting with Bit 0 (the low order bit) and working towards Bit 7. Prior to starting the bit testing, the HL register set is initialized to **404D**. This is the start of a list of addresses of routines used to process the individual interrupts. As each bit is tested and found to be false (zero), the address in HL is bumped to the next location. When an active bit is found, the registers are saved, **4547** is pushed onto the stack as a return address, and control is passed to the address pointed to by the HL register. Locations **404D - 405C** are initialized as shown below.

Figure 3.7 *Interrupt Task List Addresses*

404D	45	37	Routine Address for Interrupt Bit 0
404F	45	37	Routine Address for Interrupt Bit 1
4051	45	37	Routine Address for Interrupt Bit 2
4053	45	37	Routine Address for Interrupt Bit 3
4055	45	37	Routine Address for Interrupt Bit 4
4057	45	37	Routine Address for Interrupt Bit 5
4059	45	F7	Routine Address for Interrupt Bit 6 (disk)
405B	45	60	Routine Address for Interrupt Bit 7 (clock)

When the service routine completes, control returns to **4547**, where all registers are restored. If more bits in the status byte are on, they are tested and the appropriate processor is executed. After all status bits have been tested, control is returned to the point of interrupt with interrupts enabled.

3.4.2 Disk Interrupt Service Routine

This routine is entered by the interrupt processor whenever a disk interrupt occurs. The code is brief, it consists of the following instructions:

Figure 3.8 *Entry Routine*

```
ADDRESS      INSTRUCTION          COMMENT

 45F7        LD   A,(373CH)       get controller status
 45FA        RET                  return to interrupt
                                  processor
```

3.4.3 Clock Interrupt Service Routine

This routine is entered from the interrupt processor whenever a clock interrupt is recieved. It is used to control the execution of routines which must be executed at fixed intervals. For example, the clock maintenance code at **45AF** must be run at least once a second in order to keep accurate time. Actually, it's run once every 200 milliseconds, but more about that later. Other routines which must run on a fixed basis, if active are the TIME and TRACE subroutines.

The clock interrupt processor is entered every 25 milliseconds. It has a list of the addresses of the subroutines to be executed on a periodic basis. These subroutines are executed every time the clock interrupt processor is entered (once every 25 milliseconds), or once every 200 milliseconds. The rate at which the subroutines are called by the clock processor is controlled by their position in the address list. That list has the following format:

Figure 3.9 *Address List for Clock Interrupts*

```
ADDRESS   CONTENTS   CONTENTS  CONTENTS/DESCRIPTION

 4500       45A3       45A2      RET/not implemented
 4502       45A3       45A2      RET/not implemented
 4504       45A3       45A2      RET/not implemented
 4506       45A3       45A2      RET/not implemented
 4508       45A3       45A2      RET/not implemented
 450A       45A3       45A2      RET/not implemented
 450C       45A3       45A2      RET/not implemented
 450E       45AF       45B2      CLOCK MAINTENANCE
 4510       45A3       45A2      RET/not implemented
 4512       45A3       45A2      RET/TIME display if active
 4514       45A3       45A2      RET/not implemented
 4516       45A3       45A2      RET/TRACE display if active
```

The list contains two levels of indirection. **4500** points to **45A3**, which contains the address of the first instruction of a subroutine. The list may be divided into two parts. The first part, **4500 - 450E**, contains the addresses of routines that are executed once every 8 clock interrupts (or 200 milliseconds). Every time a clock interrupt occurs, one of the routines in this part of the list is executed. A counter in location **4040** is used to control the selection of the routine to be executed. It is incremented each time the clock interrupt processor is entered. The low order three bits of the counter (times two), are used as an index into the list for selecting the routine to be executed on any interrupt.

The second part of the list, **4510 - 4516**, contains the addresses of routines to be executed on every interrupt. They are executed unconditionally. Normally, these entries point to **45A3**, but if a TIME or TRACE command has been executed, their entries are modified to point to the appropriate subroutines. Addresses in the list starting at **4500** may be changed by two system subroutines described in section 3.4.10.

The code for the clock interrupt processor starts at **4560**. It begins by executing the subroutine pointed to (indirectly) by **4510**. This is accomplished by calling an in-line subroutine at **457B**. An index into the list at **4500** is passed as a parameter in the A register. The parameter is one half of the index to be used, since it will be multiplied by 2.

The subroutine at **457B** begins by multiplying the parameter by 2 and combining it with **4500** to form the address into the first list. This address is then saved at **45A6** for possible modification by another subroutine called through **4416**. Next, the content of the address formed is loaded into the DE register and then transferred to the IX register. Following that, the content of the formed address is loaded into the DE register (this is the second level of indirection), then moved to the HL register where control is passed to that address by an indirect jump through the HL register.

When the subroutine entered by the indirect jump completes, it returns to the caller of **457B** through a RET instruction.

Three more calls to **457B** are made using the indices 9, A and B, which result in calls to the routines pointed to by **4512**, **4514** and **4516**. After the last call to **457B**, the counter in **4040** is incremented, and the low order three bits are isolated in the A register. Control then falls into **457B**, which will form an address in the range of **4500 - 450E** based on the index. The contents of that address will be loaded and control passed to it. Return will be made to the caller of the clock interrupt processor (in this case the interrupt service processor).

3.4.4 Clock Maintenance Code

This subroutine maintains the time of day clock in seconds, minutes and hours. The time is kept in binary at locations **4041**, **4042** and **4043**.

The clock maintenance code is entered from the clock interrupt service routine once every 200 milliseconds. Since it is entered every 200 milliseconds (5 times a second), it begins by decrementing a local counter to test if it has been entered 5 times. If so, one second has elapsed and the second counter (**4041**) is incremented.

After being bumped, the second counter is tested for being equal to 60. If it is, one minute has elapsed, and the minute counter (**4042**) is incremented. Next, the minute counter is tested for being equal to 60. If it is, one hour has elapsed, and the hour counter (**4043**) is incremented.

The maintenance code begins at **45B2**. It begins by decrementing a local variable which contains a 5. This variable is used to keep count of the number of times the subroutine has been entered. On every 5th entry, one second has elapsed. If a second has not elapsed since the last entry, control returns to the caller (the clock interrupt processor). Notice that the IX register is used as a base register for this operation. This is possible because it was initialized to the origin of the subroutine by the clock interrupt processor.

If the decremented value goes to zero, it is re-initialized as 5 and preperations are made to increment the second, hour and minute counters. These preperations consist of setting a loop counter (B register) to 3, and loading the HL and DE registers with the address of the counters and a list of values representing the maximum values for each of the counters (except the hour counter).

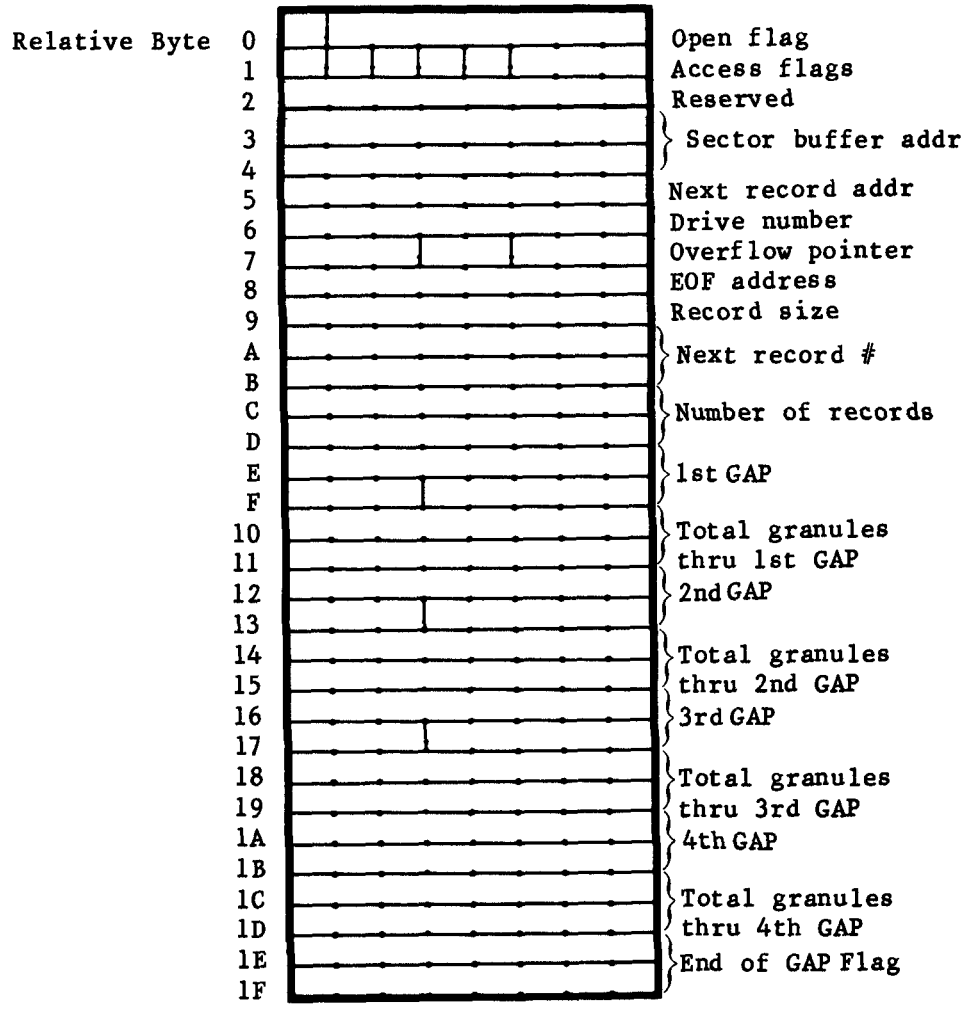
Next, a loop starting at **45C2** is entered. This loop increments a counter pointed to by the HL register and then compares the current value to a maximum value pointed to by the DE register pair. If the counter value has not reached its max, control returns to the caller (clock interrupt processor). If the maximum value has been reached, the counter is re-initialized to zero, and the entire process is repeated for the next counter. The loop terminates when three counters have been incremented or a counter is incremented without reaching its maximum limit.

3.4.5 Disk File Operations

Before a disk file can be accessed, it must be OPEN'ed, or if it is being created, an INIT call must be made. The purpose of both calls is similar. The OPEN call locates the directory entry for a file and copies pertinent information to the DCB. The INIT call creates a directory entry and initializes the DCB so a file may be written. Both calls expect a DCB address as a parameter. The DCB must contain the file name to be located or created, an optional drive specification and an optional password.

After the OPEN or INIT call, the DCB contains a precise description of the file. This description includes the drive number where the file resides, permission flags, the address of the sector buffer, the record length, the extent (length) of the file and a list of disk addresses assigned to the file. A description of the DCB is given below.

Figure 3.10 *Disk DCB Description*



After a file has been OPEN'ed it can be addressed at a record level through calls to system READ, WRITE, POSITION and REWIND subroutines. Record access will take place at a logical or a physical level depending on which was specified when the file was opened. Physical records are equivalent to one sector. Reading or writing a physical record always results in a data transfer.

Logical records are usually smaller than a sector; however, they may be the same size as sector, but no larger. In most applications where logical records are used, several records will fit into one sector. When a READ or a WRITE occurs, a logical record is moved between the sector buffer and the caller's data area. When no logical records remain in the sector buffer, or if the buffer is full (depending on the operation being performed), a physical I/O call will be automatically issued to fill or flush (write buffer to disk) the sector buffer.

Records may be accessed randomly or sequentially according to a record number in the DCB. Each call, except for REWIND or POSITION, causes the record number to be updated by one. Records are numbered from 0 to 65535.

All file accesses are accomplished through calls to system routines which perform the required function. Parameters specified are the same for all calls and include: a DCB address, a sector buffer address and the logical record length. If physical I/O is being performed, the logical record length is zero.

There are several entry points in TRSDOS for file operations. They are:

4436 - read
4439 - write
443C - write with verify
443F - rewind
4442 - position to record
4445 - backspace one record
4448 - skip to end of file

These addresses contain jumps to the specified functions. The code that actually performs the operation is located elsewhere. All of these entry points expect the same parameters, and all of them begin by testing the DCB to insure the file has been opened. Assuming that to be true, the IX register is set to the DCB address, the caller's registers are saved, the sector buffer address is saved at **430C**, the DCB address is saved at **430A** and an exit address of **48B3** is pushed onto the stack. This address is used to restore the caller's registers before returning.

READ and WRITE processing are similar enough to be discussed together. Both begin by testing for logical versus physical I/O. This test is made on a flag bit in the DCB.

Logical READs begin with a test for end of file. This is done by comparing the requested record number against the largest record number known to exist on the file. If end of file is detected, an error status is returned. Otherwise repeated calls are made to a subroutine at **47BB** to fetch the next character from the sector buffer belonging to the file. This subroutine begins by testing a flag in the DCB which indicates if a physical read is required to fill the sector buffer. If the flag is set, the next sector is read. Following the test or the read, a second subroutine at **4883** is called, which computes the address of the next available data byte by adding the current offset to the origin of the sector buffer. The calling routine (**47BB**) maintains an offset to the next available character in the sector buffer in the DCB associated with the file. When the offset goes to zero (rolls over because of overflow), a flag in the DCB is set, which causes the next physical sector to be READ.

No test for end of file is made if the operation is a logical WRITE. Like the READ routine, a subroutine is called on a byte basis, but in this case it is called to store a byte in the sector buffer. The byte store routine is located at **47DA**. It begins by testing a flag in the DCB to determine if the sector buffer is full and needs to be written to disk. If true, the sector write routine at **482D** is called to purge the current buffer. After flushing the buffer, or if the buffer was not full, the next character is stored in the sector buffer and control returns to the main loop, which starts at **478B**.

The physical READ and WRITE routines are different from their logical cousins in that they always read or write a sector. The routines they call to prepare for these transfers are used by the logical operations when a physical transfer is needed.

Before the disk driver can be called to transfer a record, the record number must be translated into an absolute track and sector number to be used as parameters to the disk drivers.

Translation from a record number (i.e., sector number), to a track and sector number consists of three steps. First, the record number must be converted to a 'relative' granule and sector number. Second, the 'relative' granule number is used to locate the granule allocation pair (GAP) in the DCB associated with the record. Third, a track and sector number are derived from the GAP located in step two. These steps can best be illustrated with the following example.

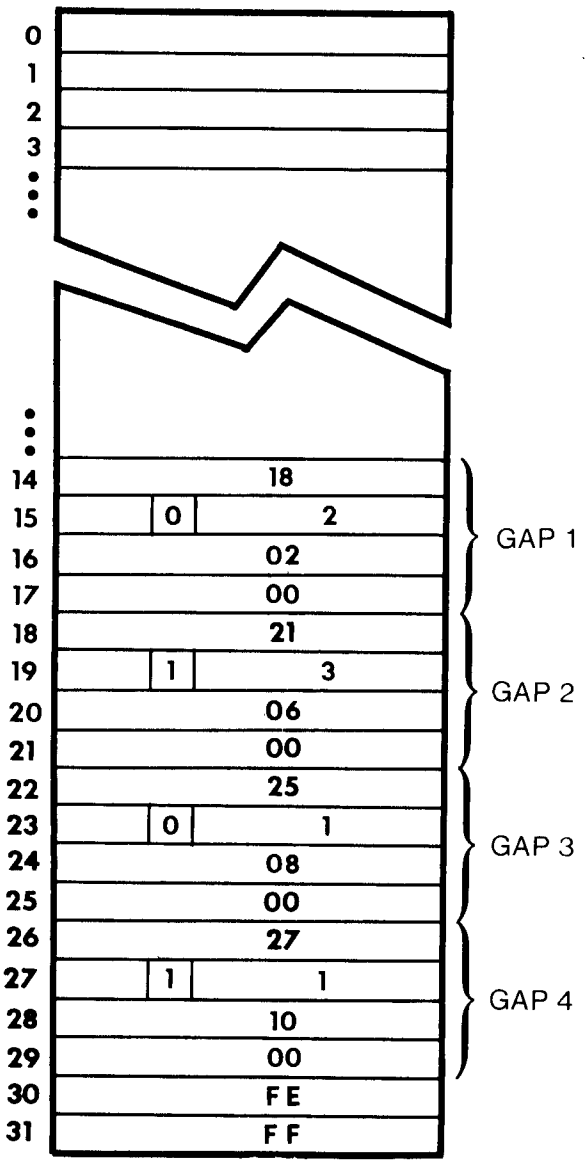
Assume we have a file containing 73 records (in this case, one record equals one sector) and we wish to read record 38. Let's make a further assumption that the following tracks and sectors are assigned to the file:

Figure 3.11a *Track and Sector Assignments*

RELATIVE GRANULE	TRACK	SECTOR	
Ø	18	Ø-4	
1	18	5-9	GAP 1
2	19	Ø-4	
3	21	5-9	
4	22	Ø-4	GAP 2
5	22	5-9	
6	23	Ø-4	
7	25	Ø-4	
8	25	5-9	GAP 3
9	27	5-9	
1Ø	28	Ø-4	GAP 4
11	3Ø	5-9	
12	32	Ø-4	GAP 1 (In overflow entry)
13	32	5-9	GAP 2 (In overflow entry)

After being OPEN'ed, the DCB for this file would appear as follows:

Figure 3.11b *File DCB*



Step one is to convert the required record number into a 'relative' granule number. This is accomplished with a modulo division by five. Five is chosen as the divisor because there are five sectors in a granule. In our example,

38 / 5 = 7 with a remainder of 3, where:

7 = 'relative' granule number and,
3 = sector offset within granule.

The 'relative' granule number is the number of the granule assigned to the file which contains the target sector. The sector offset is the number of the target sector within the granule. This computation can be verified by referring to the granule and

track assignments shown above. Record 38 would reside in 'relative' granule 7, as can be proved by totaling the sectors in granules 0 through 6.

Step two consists of searching the DCB until the GAP containing granule 7 is found. This step requires more explanation of the GAP's (as they are used in the DCB's).

GAP's in the DCB consist of four bytes. GAP's stored in the directory entries contain two bytes. The extra two bytes in the DCB GAP's are used to facilitate locating a GAP containing a particular granule number. These bytes contain the maximum relative granule number addressable by the GAP.

For example, consider a file composed of the GAP's shown above. GAP 1 contains three granules (two on track 18 and one on track 19). The largest 'relative' granule addressable using GAP 1 would be 2.

GAP 2 contains four granules. Counting the three granules in GAP 1, the largest granule addressable using GAP 2 is 6.

In step two, the granule number computed at step one is compared to the granule totals in bytes 3 and 4 of the GAP's until the GAP containing the required granule is found.

In our example, granule 7 contains the target sector. GAP 1 can be skipped because 7 is greater than 2. GAP 2 can be skipped for the same reason. GAP 3 contains the required granule number because 7 (the required granule number) is between 8 (the largest addressable granule in GAP 3) and 6 (the largest addressable granule in the previous GAP).

Step three derives the track and sector number for the required granule using bytes one and two of the GAP. In the example chosen, the derivation is simple. Granule 7 is the first granule in the GAP; therefore, the target track number is 25, and the sector number is 3.

The code which performs these steps occupies location **480C - 49B4**. A subroutine at **49B5** may be called under special circumstances.

The code appears to be quite lengthy in comparison to the explanation. The reason for this is that none of the exception conditions were discussed. For example, notice that the maximum granule number in the DCB is 10, yet the file contains 14 granules. What would happen if we wanted to read record 67 (granule 13, sector 2)? Also, there was no detailed explanation of step three, which can become somewhat involved as seen from the code at **496A - 49B4**.

The controlling code for physical reads and writes occuppies **480E** through **482C** and **483E** through **4876**, respectively. Both of the controlling routines call a subroutine at **48DC - 4926** to perform the record number to disk address translation. Ordinarily, this translation would be starightforward, but there are two cases which may complicate the processing. The first occurs when a very large file is accessed and the GAP/VALUE set owning the record is not in the DCB. The second case occurs when a file is being created and a GAP must be assigned. Both special cases, as well as the more common cases are discussed below.

The subroutine at **48DC** begins with a mod 5 division of the record number. The result is the granule (quotient), and the sector within the granule (remainder) containing the target record. Five is used as a divisor because there are five sectors to a granule. Following the division the GAP/VALUE sets in the DCB are searched until the GAP containing the computed granule is found. The loop for this search begins at **48F9** and ends at **4926**. It may be terminated for two reasons. First, the correct GAP may be found. In that case control goes to **496A**, where the track and sector number are derived. The second possibility is that the GAP may not be found. That may be due to one of two reasons. First, the GAP being sought may exist in an oveflow entry which has not been read yet; second, if the file is being written for the first time, GAP's must be assigned by the disk space allocator. In either case, if the GAP owning the granule is not found, control will go to **49B5**.

At **49B5** the primary directory entry for the file is read so that a test for overflow entries can be made. If there are no oveflow entries, control goes to **4A00**, where a GAP is assigned. Since end of file tests were performed earlier, it is safe to assume a WRITE operation is in progress.

If overflow entries exist, they are read and searched for the GAP owning the granule number computed from the record number. All overflow entries are read until one containing the desired GAP is found. The search is performed in the subroutine at **49B5**. An exit from **49B5** is not taken until the GAP being sought has been found, or one has been assigned if the operation is a WRITE.

Following the call to **48DC**, the read/write next sector flag is zeroed, the record number in the DCB is incremented and the disk driver is called to perform the physical transfer.

3.4.6 Disk Driver

The disk driver resides at **4671 - 46DC**. It is generalized for READ and WRITE operations only. No other operations can be processed by the driver. The driver transfers one sector per call. Multi-sector calls are not supported. It is entered through calls to one of three subroutines, which setup a parameter list before calling the driver. Upon exit from the driver, control returns to the caller of the subroutine used to call the driver. The addresses of the subroutines are:

46DD (read a sector)
46E6 (write a sector)
46F3 (reads sector but does not transfer data).

The calling sequence to the driver is as follows:

Figure 3.12 *Driver Calling Sequence*

```
LD    A,OPCODE      READ/WRITE op code
LD    C,UNIT        unit #
LD    D,TRACK       track #
LD    E,SECTOR      sector #
LD    BC,BUFFER     buffer address
```

Listing Continued . . .

. . . Continued Listing

```
CALL 4671H          call driver
DEFB COUNT          retry count
DEFB BIAS           bias to be added to status bit
DEFW INSTR          LD  A,(BC)/LD  (DE),A
```

Subroutine calls are used to select the unit and initiate all physical motion except the data transfer. Those subroutines may call other subroutines to perform part of the required function. Main subroutines called from the disk driver are: select unit and position to track and sector (**4647**), wait for controller ready (**4455**) and error recovery -- issue RESTORE and retry operations (**4C9B**). Subroutines called from these subroutines are: get disk status (**45F3**), restore head (**4658**) and build unit select mask (**4639**).

The disk driver begins by saving the op-code in location **469A**. Next, the parameters are retrieved, and the registers are set up for the data transfer operation. Following that, the unit is selected and positioned to the desired track and sector. The subroutine used for this begins at **4600** and ends at **4638**.

It begins by waiting for the controller to become not busy. Next, the unit number is saved in **4803**; then the track for the currently selected unit is fetched from the controller and saved in the track history table beginning at **430A**. This table is indexed by unit, it contains the last track addressed on each unit. Following that, a unit select mask is constructed by a call to **4639**. This mask is derived from the unit number by synthesizing a SET instruction using the drive number as the bit number to be set. The mask is then stored in **37E1**, causing the drive to be selected.

The unit select mask for the current unit is saved in **4309** for future reference. After selecting the unit, the drive is tested to see if it is ready, and, if so, the subroutine exits back to the driver. Otherwise, a delay loop is executed before returning to the driver.

Following the call to **4647**, the driver calls **4455** to wait for drive ready. Upon return, interrupts are disabled, the command is sent to the controller and a test is made to determine if the sector buffer overlays the last sector buffer used for a system overlay. If so, **430E** is cleared to indicate the overlay area has been used and that the next overlay to be used must be loaded from disk.

Next, the main loop for the data transfer is entered. The loop is short and starts with a sub-loop that tests for data available or drive not ready. If data is available, or the controller is able to accecpt the next data byte, the transfer is made, the buffer address is bumped, and control returns to the sub-loop described above. Eventually, the drive will become not ready (after a sector has been transferred), and control will pass to **46AD**, where the status will be examined and the controller is shut down with a force interrupt command.

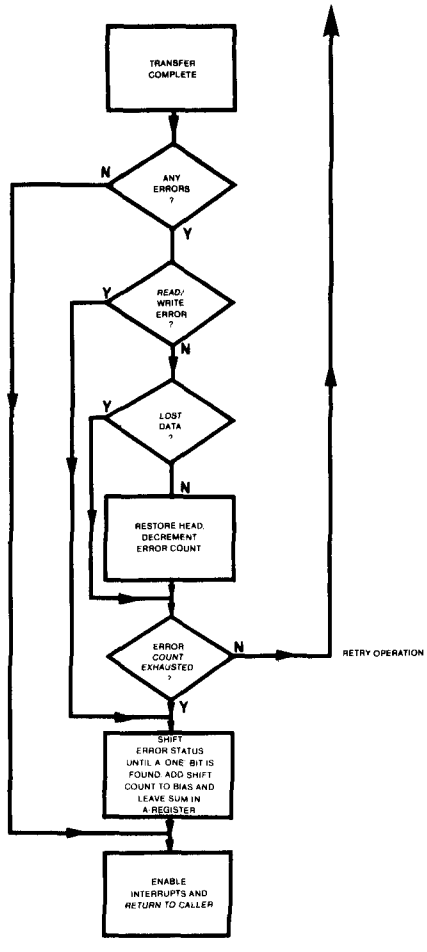
The status test starts with a quick test for any errors. If none are found, interrupts are enabled and the driver returns to the caller.

If any kind of error occurred, a second test for a read/write fault or lost data is performed. If neither was the case, 'unit not ready' or 'record not found' is assumed. In this case, interrupts are enabled, the heads are restored and the error retry is decremented. If the error count is not exhausted, the operation is retried.

If the status indicated lost data, the error count is decremented, and if it has not been exhausted, the operation is retried.

When the error count is exhausted or a read/write fault is detected, the number of the first non-zero bit in the status (starting from bit 0) is added to the bias value passed in the parameter list. The calling program will compare this value against a constant to determine if the operation terminated without error. The sum is left in the A register, interrupts are enabled, and control is returned to the caller. The flowchart below illustrates this processing

Figure 3.13 *Disk Error Recovery Flow Chart*



3.4.7 Disk Space Management

TRSDOS supports 35-track, single-sided, single-density disk drives. Up to four drives can be connected to a system. A fixed sector size of 256 bytes is used. Each track contains 10 sectors. Tracks are formatted with the sectors interleaved in the order 0, 5, 1, 6, 2, ...

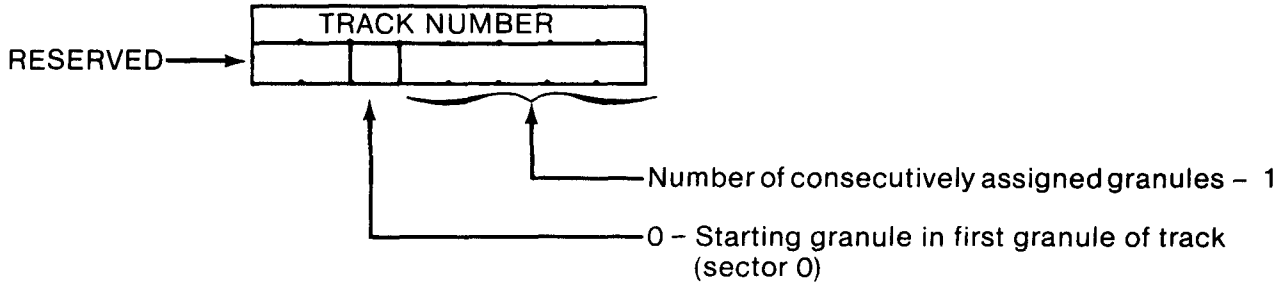
A diskette can contain up to 64 files. There is no limit to the size of a file except that it cannot overflow a diskette.

A diskette is divided into units called granules. A granule is five consecutive sectors. Each track contains two granules, one starting on sector zero and ending on sector four, the other starting on sector five and ending on sector nine.

A bit map is used to keep track of the available granules. The bit map is kept on track 11, sector 0 and is called the GAT (GRANULE ALLOCATION TABLE) sector. It occuppies the first 80 bytes of the sector. Since TRSDOS supports only 35 tracks, the last 45 bytes of the bit map are unused. Each byte represents one track. Bits 2 through 7 are not used (they are initialized to binary ones). Bits 0 and 1 represent the first and second granules, respectively, for the track associated with the index into the bit map. A zero means the granule is available, a one means the granule has been assigned. A complete description of the GAT sector may be found in Section 2.5.

A list of granules assigned to a file is kept in both the directory entry for the file and in the DCB (after the file has been OPEN'ed). The list consists of a set of Granule Allocation Pairs, or GAP's. A description of a GAP is given below.

Figure 3.14 *Granule Allocation Pair Description*



The first byte contains the beginning track number. The second byte contains the number of consecutively assigned granules minus one plus a flag indicating the starting granule sector number. There is no limit to the number of GAP's that can be assigned to a file. However, a directory file entry will hold up to 5, and a DCB up to 4. If a file has more than 5 GAP's, an overflow directory entry is created to hold the surplus GAP's.

Each GAP can represent a maximum of 32 granules because the width of the count field is 5 bits. Whenever there is a break in the number of consecutive granules, a new GAP is created.

When a file is initially created, the GAP's are maintained in the DCB until the file is CLOSE'd, at which time they are copied to the directory. If, however, while updating a file the GAP area of the DCB becomes full, a portion of the GAP's will be transferred to the directory.

The use of the GAP's in the DCB is slightly different than in the directory. Referring to Fig. 3.10, notice that there is a two-byte value following each GAP. The value is the total number of granules addressable through the preceeding GAP. The system uses this value to locate the GAP to use for address calculations. When accessing a record, the record number is translated into a granule number. This granule number is then compared to the values following the GAP's in the DCB until the value nearest to it is found. The GAP belonging to the nearest value is used for the disk address calculation. For example, if the first GAP was followed by a value of 250 (hex.), and record 10 (hex.) was being accessed, then the first GAP could be used for computing the absolute track and sector number. If, on the other hand record 350 (hex.) was being accessed, the search would continue until the first value greater than 350 was found. Then, the preceeding GAP would be be used. For more details relating to record translation using GAP's, see Section 3.4.5.

Disk space is allocated by the code from **49F1** through **4B34**. This is a broad region of code which may be subdivided into eight seperate areas. Two of those areas (one at **4A2A - 4A68**, and the other at **4A6E - 4ABF**) actually scan the bit map and assign granules. The remaining areas contain suppport code to: read the GAT sector (**4AF0 - 4B02**), write the updated GAT sector after a granule has been assigned (**4B03 - 4B1DD**), read the directory entry for the current file (**4AC1 - 4AD5**), build an overflow directory entry (**4A6B - 4A6D**), write the updated directory entry (**4AD6 - 4AEF**) and compute the track and sector number for a directory entry (**4B1E - 4B34**).

The codes of interest here are the two areas that allocate granules. Both have a common section (**4A00 - 4A28**) which is executed before either is entered. The common section reads the GAT sector containing the bit map into the buffer at **4D00** and then determines if an initial granule assignment is being made or a granule is being added to an existing GAP.

There are several reasons for distinguishing between the two cases, but the most obvious one is that for an initial granule assignment, any granule can be assigned; whereas, when assigning granules to an existing file, it makes sense to try to assign the granule following the last one assigned to the file. For example, if a file had three granules assigned starting at track 10, sector 0, and the file was being extended, it would be more efficient to assign the granule for track 11, sector 5 than any other granule. First, this would allow the GAP for the existing granules to be used; otherwise, a new one must be created. Second, there is an advantage in using granules close to those already assigned because head movement is minimized, and if the granules are consecutive, there is no unnecessary motion.

The code at the tail end of the common section terminates by testing if the first GAP for the file is being created. If it is, control goes to **4A6E**, where an available granule is found and a GAP entry is created. If a GAP already exists, its count field is tested for being full (20 hex.), and if full, control goes to **4A54**.

At **4A54**, a second test is made to determine if the end of the GAP space in the DCB has been reached. If not, control goes to **4A6E**. There, a granule will be assigned, and a GAP entry will be added to the DCB.

If the DCB is full (all GAP entries have been used), the GAT and directory sectors are updated on disk; then an overflow directory entry is created (call to **4A6B**), and processing resumes at **49B5**. Recall that the maximum number of GAP's that can be carried in a directory entry is 5. That is also the maximum that can be carried in a DCB. When a DCB is full, the GAP section of the directory entry must be full as well. Therefore, an overflow directory entry must be created.

Assuming none of the above cases are true (we are trying to add a granule to an existing GAP), the code from **4A44 - 4A52** is executed. This code begins by testing the last granule assigned. If it was the last physical granule (**4A44 - 4A4B**), a new GAP must be created, and control passes to **4A54**. Otherwise, the subroutine at **4B5D** is called to determine if the next consecutive granule is available. If it is, control goes to **4A9C**, where it is marked as assigned, and the granule count in the DCB is incremented. If the granule is not available, control goes to **4A54** (which has already been described) to create a new GAP.

The code starting at **46AE** scans the bit map, looking for an available granule. This code may be entered when the first granule of a GAP is being assigned, and consequently, it must be able to construct the GAP entry as well as assign granules. The map may be scanned twice because on entry, the HL register pair contains a starting position in the map for the scan. Since this may not always be the start of the map, the scan must be performed twice if no granule is found on the first pass.

When an available granule is found, control goes to **4A89**, where a number of interesting things happen. First, the count of granules assigned is initialized to minus one in case an initial granule assignment is being made. This count will be incremented by one for each granule assigned, and the result will be copied to the second byte of the GAP if necessary. Second, a loop from **4A8F** to **4A9A** is executed to determine exactly which granule (the first or the second) of the track is available. This is important, because bit 5 in the second GAP byte must be set to indicate if the first granule begins on sector 0, or sector 5.

As a part of this loop, a subroutine at **4B5D** is called to build a BIT instruction which will be used to determine the exact bit (granule 0 or granule 1) that is available. After the exact bit for the granule to be assigned has been found, control goes to **4A9C**, where a subroutine at **4639** is called to flag the granule as unavailable in the GAT. Next, a test at **4AA3 - 4AA5** determines if this is an initial granule assignment and, hence, the GAP must be created, or if a granule is being added to an existing GAP. If the latter, control goes to **4AA9**, skipping over the code which copies the track owning the granule to the DCB.

Finally, a test is made to determine if the required number of granules has been assigned. Usually, this number is two; however, in some cases it is one.

When the required number has been assigned, the updated GAT bit map is written back to the diskette it was read from, the directory entry for the file is rewritten (it has been updated) and control returns to the calling routine.

3.4.8 File Loader

TRSDOS consists of a core resident nucleus plus various system overlays and utility programs. The system overlays are loaded into a special region within the nucleus (**4E00 - 51FF**), while the utilities are loaded at **5200** and above. User-written assembly programs can be loaded and executed as utility programs.

Both the overlays and the utilities are kept on the disk as named files. They are stored in assembler object format with embedded control codes specifying the load addresses, the type of data (text or code), and the beginning execution addresses. The format for assembler object files is given below.

Figure 3.15 *Object Code Format*

```
                    Assembler Object Code Format

The general form of an object file is:

     (Control Code) (Count) (Load Address) (Load Data)

In some cases the count field or load address may be omitted.

            Control Code:    01  (data to be loaded follows)
            Count        :   XX  (count of bytes to load, 0 = 256)
            Load Address:    XX  (load address in LSB/MSB order)
                             XX
            Load Data    :   XX
                             XX
                             .

            Control Code:    02  (beginning execution address follows)
                             XX  (this byte is to be discarded)
            Address      :   XX  (execution address in
                             XX  (LSB/MSB order)


            Control Code: 03 - 05  (following data is to be skipped)
            Count        :   XX  (count of bytes to skip)
            Skip Data    :   XX  (this data is to be skipped)
                             XX
```

A loader (**4C39 - 4C9A**) is used to load any assembler object file. The loader is called from different points within the nucleus, depending on whether an overlay or a utility is being loaded.

Utilities are loaded through a file call to **4430**, which is routed to **4C16**. Overlays are loaded using a special RST function which gets routed to **4BA2**. The overlay loader is discussed in the next section.

The file loader (**4C16 - 4C35**) expects the registers to be loaded for an OPEN call. The B and HL registers need not be set, as they will be initialized to a **00** and **4200**, respectively. After the registers have been initialized, an OPEN call is made against the DCB, passed as an argument.

If the OPEN is successfull, the permission flags are tested to verify execute access on the file before control goes to the loader.

The loader occupies locations **4C39** through **4C9A**. Upon entry, the DCB address for the load file is saved at **4C8D** and the E register is set to minus one.

It is assumed that the DE register set points to the sector buffer for the load file. The buffer is assumed to start on an address which is an even multiple of 256. Thus, the D register contains the page number, and the E register is used as an index into the page (buffer).

The main loop extends from **4C40** to **4C87**. Two subroutines are used from the main loop. One is called to fetch the next byte from the sector buffer.

This subroutine (**4C89 - 4C9A**), increments the address in DE and fetches the next data byte from the buffer. If the buffer is exhausted (E register equals zero), the next sector is read from the file and the DE register is reinitialized to **4200**.

A second subroutine is called to read the next sector when, under special circumnstances, the buffer becomes exhausted without calling the fetch next byte routine. This situation arises when records in the load file are being skipped.

The main loop calls the next byte routine to fetch the control byte, the byte count and the load address (two bytes) for the data. The control byte is tested to insure it is within range (< **1F**). If not, an error status of **62** is returned to the caller.

Assuming the control code is in range, the byte count and the LSB of load addresses are fetched. At this point, the control byte is tested, and if found to be a 1, control goes to **4C6C**. Here the number of bytes specified in the control area is transferred from the sector buffer to the load address. Following each transfer, a test is performed to insure the data was successfully stored and that the sector buffer has not been exhausted. If the data was not stored successfully, control goes to **4C80**, where a test is made to determine if an error occurred on the fetch instruction (error code **24**), or on the store instruction (error code **23**). After the cause of the error is determined, the appropriate error message is displayed.

When the byte count is exhausted, control returns to **4C40**, where the next control byte is read.

If the control byte does not equal 1, it is compared with a 2. If it is 2, the next byte is fetched and discarded so that the transfer address (starting execution address) may be loaded into the HL register.

If the control byte is not a 1 or a 2, control goes to **4C68**. The code at this address skips forward to the next control byte in the sector buffer. When the buffer pointer is positioned to the next control byte, control goes to the start of the load loop (**4C40**).

3.4.9 Overlay Loader

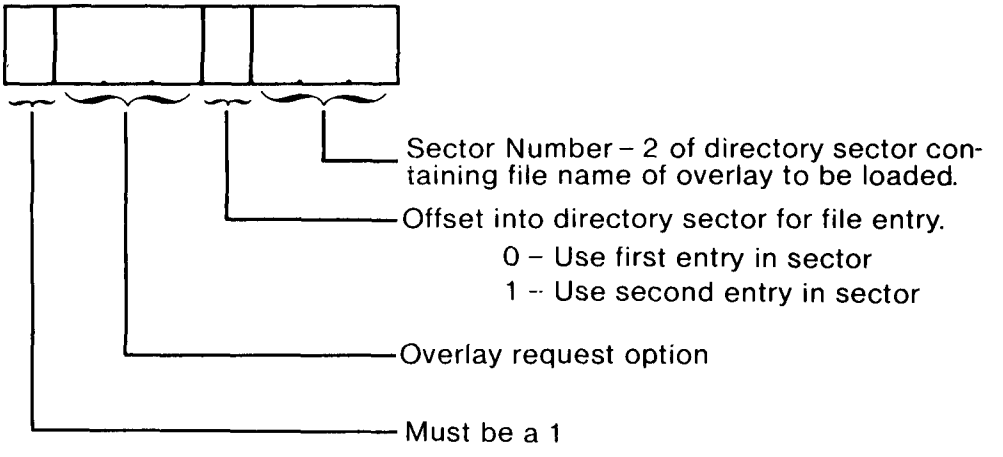
The overlay loader occupies locations **4BA2** through **4C05**. It is used to load and execute system overlays. The overlay loader is similar to the file loader, but there are several important differences between the two.

The file loader expects the DE register to contain the address of a DCB for the file to be loaded as a parameter. The overlay loader is given the number of the directory sector containing the name of the file to be loaded (the file name must occupy the first or second entry in the sector).

Programs loaded as files, as opposed to overlays, cannot have parameters passed through registers. They must examine the load call in the command line buffer to locate any parameters. Overlays are entered with a parameter byte in the A register. In addition, overlays return a status to the caller via the A and STATUS registers. And finally, the overlay loader does not reload an overlay if it is already core resident.

Overlays are loaded using the following calling sequence:

Figure 3.16 *Overlay Request Code Description*



The overlay loader begins by disposing of the return address on the stack generated by the RST instruction. The return address of the caller of RST subroutine is then loaded into the HL register set.

Next, the parameter in the A register is tested. If bit 7 is zero, it is an invalid parameter, and control is returned to the caller. Assuming a valid parameter was passed, the return address is saved on the stack, and the incoming parameter is compared to the previous parameter. If they are the same, control goes to **4BC0**.

At **4BC0** the overlay is entered with the parameter in the A register. Since overlays may be re-executed without being reloaded, they must be written so they are re-enterent. This means any counters must be initialized by load and store instructions rather than being defined as constants via DEFB or DEFW pseudo-ops. For example, consider the following sequence of code:

Figure 3.17 *Non Re-enterent Code Example*

```
LOOP    LD      A,(COUNT) ;load A with whatever
                           is in "Count" address
        DEC     A
        LD      (COUNT),A
        JR      Z,END
            .
            .
            .
        JR      LOOP
        END
            .
            .
COUNT   DEFB    1Ø
        END
```

This code could only be executed once, because the value used to control the loop (COUNT) is initialized when the program is loaded. In order for this code to be re-executable without being reloaded, it would need to be changed as follows:

Figure 3.18 *Re-executable Code Example*

```
        LD      A,1Ø
        LD      (COUNT),A ;COUNT NOW ALWAYS
                          CONTAINS SAME VALUE
LOOP    LD      A,(COUNT)
        DEC     A
        LD      (COUNT),A
        JR      Z,OUT
            .
            .
            .
        JR      LOOP
        OUT
                    .
                    .
                    .
COUNT   DEFB    Ø
        END
```

If the lower 4 bits of the overlay request code do not match the same bits from the previous request, a new overlay must be loaded.

The code from **4BB6** - **4BEC** is used for the load operation. It begins by testing bit 3 of the request to determine which entry in the directory sector contains the name of the file to be loaded. Depending on the results of the test, the index into the sector buffer (contents of the A register) may be adjusted. In any case, the sector buffer index is copied into the B register for use by the subroutine at **4B1E**, which will be called indirectly during subsequent processing. Following is the code used for this test and adjustment:

Figure 3.19 *SYS0 Code From 4BB6 to 4BC0*

```
4BB6 D5          PUSH    DE          Save caller's DE
4BB7 C5          PUSH    BC          and BC
4BB8 E60F        AND     0FH         Isolate directory sector number
4BBA FE08        CP      08H         Test for 1st or 2nd entry in sector
4BBC 3802        JR      C,4BC0H     Jump if 2st entry to be used
4BBE C618        ADD     A,18H       Else get new sec no. - file offset
4BC0 47          LD      B,A         Move to B. Will be used for
                                     computing buffer addr by 4B1E
```

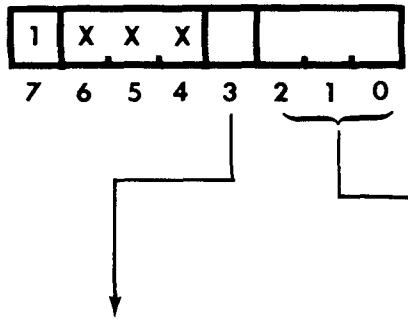
After the sector buffer address for the filename has been computed, the short system DCB is initialized to read the file name. Then, the directory sector is read through a subroutine call to **4AC1**.

The subroutine at **4AC1** saves the BC and DE registers before calling another subroutine at **4B1E** to return the track number for the directory.

The subroutine at **4B1E** in turn calls a subroutine at **4B55**, which uses a table look-up to locate the directory track number for the unit speciffied in the C register. When control returns to **4B21** from **4B55**, the directory sector number is tested to insure it lies between 0 and 8, and if so, it is adjusted to the true sector number. A sector number greater than 8 is considered an error, and a non-zero status (error status) is returned to the caller of **4B1E**.

At this point, the low 4 bits of the request code would have been translated as follows.

Figure 3.20 *Finding the Sector/Buffer Location of System Overlays*



Address of File Entry in Sector Buffer		Bit 3 Value: 0	Bit 3 Value: 1	Sector No.	Bits 0 - 2
(HL)	=	4200	4220	(E) = 2	000
(HL)	=	4200	4220	(E) = 3	001
(HL)	=	4200	4220	(E) = 4	010
(HL)	=	4200	4220	(E) = 5	011
(HL)	=	4200	4220	(E) = 6	100
(HL)	=	4200	4220	(E) = 7	101
(HL)	=	4200	4220	(E) = 8	110
(HL)	=	4200	4220	(E) = 9	111

Control will eventually return to **4AC6**, where a test for error conditions (non-zero status) is performed. Assuming no errors occurred, the address of the required file in the sector buffer is saved on the stack, a sector buffer address of **4200** is loaded in the HL register pair, and the subroutine at **4B35** is called to read the required directory sector. After the read completes, the HL, DE and BC registers are restored, and control returns to **34BD**.

At **4BD1**, a branch to **4409** is taken if any error occurred during the read operation. If no error occurred, the first GAP (starting tracking number and number of granules) is fetched from the file entry in the directory sector and transferred to the system DCB. The following code is used for this operation:

Figure 3.21 *SYS0 Code From 4BD1 to 4BDC*

```
4BD1 C20944      JP      NZ,4409H      Jump if error during read
4BD4 7D          LD      A,L           LSB of file entry addr in buffer
4BD5 C616        ADD     A,16H         Add offset to position to first GAP
4BD7 6F          LD      L,A           Form GAP address in HL
4BD8 7E          LD      A,(HL)        Fetch track number for overlay
4BD9 2C          INC     L             Bump to 2nd GAP byte
4BDA 66          LD      H,(HL)        Fetch sect. offset/granules assigned
4BDB 6F          LD      L,A           HL=1st GAP for overlay
4BDC 22AE44      LD      (44AEH),HL    Save as 1st GAP in short system DCB
```

After initializing the DCB, the system loader at **4C39** is called to load the file.

Here the GAP for the file is moved to the system DCB (the same one used to read the directory sector), the system DCB address is loaded into the DE register, and the system loader at **4C39** is called to load the file. For a description of the system loader, see Section 3.4.8.

When control returns from the system loader, the transfer address in the HL register pair is saved as the address of a CALL instruction at **4BFB**, which will eventually be used to enter the overlay. Next, the DEBUG flag is saved in local storage so it can be restored after the overlay has been executed; then it is zeroed. Following that, the parameter value is loaded into the A register, and the overlay is entered through a CALL instruction. Upon return, the DEBUG flag is restored, and control returns to the overlay caller. None of the registers are preserved.

3.4.10 Trace Display

The TRACE routine displays the PC register on the top row of the video each time an interrupt occurs. This can be very useful for debugging programs, particularly those that take wild branches or get stuck in a loop at an unknown address.

The TRACE command is recognized and processed by SYS1/SYS. No display action is taken when the command is processed. Instead, one of the interrupt address lists in the nucleus is updated so that the TRACE routine in the nucleus

will be executed on every clock interrupt (once every 25 milliseconds). Before going on, we will describe the nucleus routine called to update the address list.

Recall from Sections 3.4.1 and 3.4.4 the two address lists used at interrupt time. One (**404D - 405C**, called the INTERRUPT SERVICE list), contains the addresses of routines to be executed for specific interrupt conditions, such as clock, disk or communications interrupts. The other (**4500 - 4515**, called the INTERRUPT ACTION list) contains the addresses of routines to be executed during each clock interrupt. Although there is room for 12 addresses in the ACTION list, only those pointed to by the last four entries (**4510 - 4516**), are executed during clock interrupt processing.

A service routine in the nucleus (**4593 - 45A2**) is used to update the INTERRUPT ACTION list. There are two entry points in this routine. Both of them expect an index indicating the position to be updated. The index is passed in the B register. One (**4413**) is called to store the address of a RET instruction at the specified position in the list. The other (**4410**) is called to store the address, contained in the DE register set, in the list at the position specified by the B register.

When SYS1 processes a TRACE,ON command, a call to **4410** is made to update the INTERRUPT ACTION list. The address of the TRACE display routine (**4CDB**) in the nucleus is stored in the ACTION list at the 16th position (**4516**). Whenever a clock interrupt occurs, control will eventually pass to **4CDB** to display the address of the instruction being executed when the interrupt occurred.

TRACE,OFF commands are also processed by SYS1. In their case, **4413** is called to store the address of a RET instruction at the 16th entry of the ACTION list.

Routines entered via the ACTION list are entered through two levels of indirection. The addresses in the ACTION list are the locations of yet another pointer to the routine to be executed. There does not seem to be any reason for the second level of indirection, except for the way in which the ACTION routines were implemented.

The TRACE display routine at **4CDB** begins by computing the stack address for the interrupt address. This amounts to backspacing the system stack pointer 12 locations (except that the stack pointer is not actually changed). The address of the backspaced pointer is kept in the HL register set. It points to the return address (the address of the instruction being executed when the interrupt occurred). Next, the return address is loaded into the DE register pair, and the HL register set is loaded with the screen address where the PC register will be displayed. Following that, the return address is converted to its ASCII equivalent and stored (displayed on the video) at the address in HL, and then control is returned to the caller (in this case, **4574** in the clock interrupt service routine).

The code for TRACE occuppies **4CD9** through **4CFF**. **4CD9** contains the address of the first instruction in the TRACE routine. Remember, it is entered indirectly. Starting at **4CDB** is the code that computes the stack address for the interrupt address. **4CDF** through **4CE1** load that address into the DE register,

and the code at **4CE2** initializes the HL register with the video address.

The code from **4CEA** through **4CFF** converts the value in the A register into its ASCII equivalent and stores the result (two characters) at the address specified by the HL register. This code is CALL'ed from **4CE6** to convert and display the most significant byte of the interrupt address, and is entered directly by falling through from **4CE9** to do the same with the least significant byte of the address.

This code begins by saving the value to be converted on the stack. This step is necessary, as we shall see from the remainder of the processing. Following the save instruction, the A register is shifted right 4 bits. This gets rid of the rightmost digit of the two digits to be converted. As a result, the A register is left with a single, 4-bit hex value which needs to be converted to ASCII.

The ASCII conversion is performed by a CALL to **4CF3**. Like the code from **4CEA - 4CFF**, this subroutine is entered by a CALL for one value, and then it is entered directly from above for the the second value to be converted.

The subroutine at **4CF3** begins by isolating the value to be converted to hex with a 4-bit mask (see **4CF3**). Next, a **30** is added to the 4-bit value, yielding the ASCII equivalent for the number. The result must be tested for values greater than 10, since there is no direct mapping from hex to ASCII for the values **A - F**. If the hex character was an **A - F**, then it must be converted from a **3A - 3F** to a **41 - 46**. This entire operation is performed by the code from **4CF5 - 4CFB**. After the conversion, the ASCII equivalent of the hex value is displayed at the address specified in the HL register, the HL register is bumped and control returns to the caller.

3.4.11 Clock Display

The clock routine displays the current time on the top line of the video. The clock routine is entered from the clock interrupt processor every 200 milliseconds if a CLOCK,ON command has been executed.

Since it is entered every 200 milliseconds and the time needs updating only once a second (1000 milliseconds), the display routine has a counter to tell when it has been entered 5 times.

The code for the display occuppies **4CAC - 4CD0**. It begins by decrementing a local counter that is initialized as 5. If the result is non-zero, control returns immediately to the clock interrupt routine. When the counter goes to zero, it is re-initialized as 5, the current time is converted to ASCII and displayed.

The conversion and display loop begins at **4CBC**. This loop assumes HL contains the display address, DE the ending address of the locations containing the time in binary, the C register an ASCII :, and the B register a loop count.

Most of the loop is devoted to a binary to decimal-ASCII conversion. The conversion is simple enough because the values to be converted must be less than 60. A binary value is divided by 10 repeatedly until the quotient is less than 10. The remainder of the division is the least significant digit, and the quotient is the most significant digit.

This conversion occuppies locations **4CBE - 4CC7**. A time value from the address given in the DE register is fetched into the A register and divided by 10 by means of a compound subtraction. As the subtraction takes place, the quotient is accumulated by counting the number of subtractions as an ASCII value in the address pointed to by the HL register. This location has been initialized to a **2F**, so that each addition yields the ASCII value for a decimal digit starting with 0, 1, and so on.

When the subtraction loop completes, the A register contains the negative remainder, which is converted to a decimal-ASCII value by the addition of a **3A**. The sum is then stored at the address next to the quotient. Next, the B register is decremented, and if zero, control is returned to the caller (clock interrupt processor). If the B register is non-zero, the address in the HL is incremented, and an ASCII ':' is stored there. The HL address is incremented again, and the conversion loop is re-entered to convert the next value.

3.4.12 Keyboard Driver

During system initialization, the address for the keyboard driver is replaced with the address of another driver which is resident in the nucleus. This driver is used in place of the ROM driver because it has better 'debounce' processing.

Debounce is the elimination of unintentional repeating of a character. It is a problem on the Model I systems because the keyboard driver may be called to scan the keyboard repeatedly, and because of the keyboard used, the last character processed may still be active. The problem varies with the typist. Some have it worse than others because they take longer to lift their fingers from the keys. If a keyboard strobe occurs while the last active key is being released, the character will be repeated.

The LEVEL II keyboard driver occupies locations **03E3** thru **0457**. It uses a translation table located at **0050 - 005F** in ROM, a DCB at addresses **4015 - 401A**, and a row history buffer at **403D - 403C**.

The keyboard is memory-mapped into addresses **3801, 3802, 3804, 3808, 3810, 3820, 3840** and **3880**. Whenever a key is struck, a bit in a byte at one of those addresses is turned on. When the word is read (fetched) by the driver, it is reset to zero.

The row history buffer is used to keep track of the last active key which was detected. This is an important point. When the driver is entered, the keyboard memory addresses are read and compared to their previous values recorded in the row history buffer. After the comparison, the new value replaces the previous one. If a difference is detected, the active row is processed. A key, which is held down and never released, is not detected on successive calls to the driver because of row history buffer. Every time the active row is compared to the history buffer, an exact match is found, and no processing takes place. This is the reason that keystrokes cannot be repeated by holding down a key.

The TRSDOS keyboard driver begins at **43D8**. Upon entry the driver tests for an active key. If none is found, the driver returns to the caller. It is the responsibility of the caller to make repetitious calls in order to construct a complete record.

When an active row is found, the debounce routine is called. This subroutine delays for about 40 milliseconds before retesting the active row. If the row is inactive after retest, the first detection is considered invalidf, and control is immediately given back to the caller. If it is still active, the row index and the 'on' bit position within the row are translated into an upper case ASCII value by means of a computation or through a table look-up, depending on the row number.

notes

4

4.0 SYS1/SYS

We can think of SYS0 as the 'heart' of TRSDOS because it provides the basic functions that the rest of the system needs in order to function. Using a similiar analogy, SYS1 can be considered the 'brains' of TRSDOS because it scans all command input lines and decides what action to perform. Consequently, SYS1 is called the command line intrepreter.

Command lines must be distinguished from input lines for applications programs. For example, after the system is initialized (IPL'd), SYS1 is loaded to post a read to the keyboard for a command line. That means the next line read from the keyboard is assumed to be a system command of some sort which is to be processed by SYS1.

By contrast, if a BASIC program is executed, and it posts a read to the keyboard, the response would be processed by the BASIC program as data rather than by SYS1 as a command line. The point is that not all data read through the keyboard is processed by SYS1.

When the operating system has control of the machine because there is nothing for it to do (it is waiting for the next command), SYS1 is the part of the system that is active.

As soon as a line is entered on the keyboard (return key is struck), SYS1 examines the line to determine if it is:

1. A TRSDOS command such as DATE, TIME, DIR, etc.

2. The name of a file that exists in the directory. The file name in the directory must have a suffix 'CMD'; however, the command line need not specify any suffix.

If a TRSDOS command is detected, it is processed by SYS1 or SYS6, depending on the command. System utilities, such as BACKUP or BASIC, are treated the same as program file loads described below.

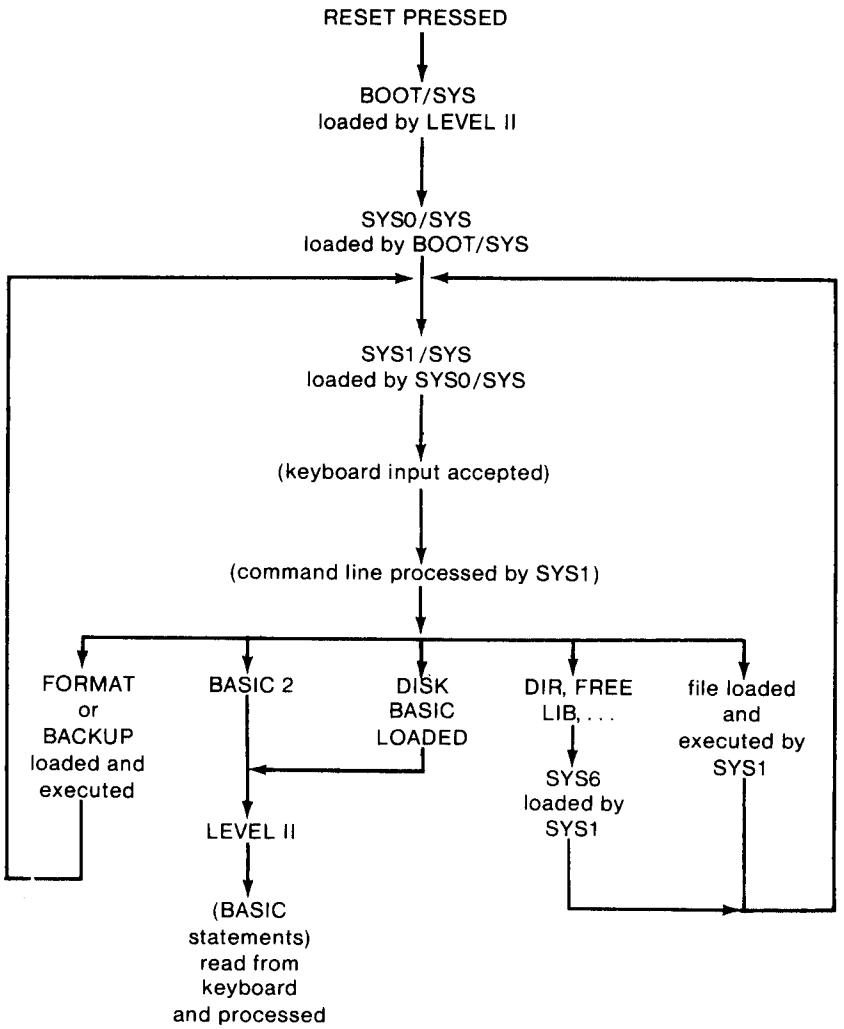
Commands not recognized as system commands are assumed to be a request to load and execute a file name that matches the command. Thus, a command of XXX, for example, is considered to be a request to load and execute the file XXX/CMD. If the file does not exist, the messsage FILE NOT FOUND is displayed, and another read for a command is posted to the keyboard.

If the file exists, but it is not a binary load module, the messsage FILE LOAD ERROR would be displayed, and another keyboard read would be posted.

If XXX/CMD exists as a loadable binary module, it would be loaded, and control would pass from SYS1 to XXX. When XXX had completed processing, it would jump to **402D** in the nucleus. This would result in SYS1 being reloaded and executed. Naturally, it would begin by posting a read for the next command, and the entire cycle would be repeated.

A diagram illustrating SYS1 processing in relation to command execution is shown below.

Figure 4.1 *Command Processing Flow*



There are six entry points into SYS1. Three of them interpret a command line, and three perform functions used by SYS6, as well as SYS1. The options parsed in the A register for entry points are described below:

Figure 4.2 *Entry Point Options*

SYS1 OPTIONS	Function
10	Read next command from keyboard and process.
20	Same as above.
30	Intrepret command in command line buffer.
40	Move string to DCB and validate as a filename
50	Add suffix to command.
60	Parse parameter list.

SYS1 begins by isolating the options field of the request code and branching to the corresponding entry point. The following code is used for this operation:

Figure 4.3 *SYS1 Code From 4E00 to 4E1E*

```
4E00 E670      AND     70H          Isolate option
4E02 FE10      CP      10H          Test for option 10
4E04 CA1F4E    JP      Z,4E1FH      Read next command and process
4E07 FE20      CP      20H          Test for option 20
4E09 2814      JR      Z,4E1FH      Read next command and process
4E0B FE30      CP      30H          Test for option 30
4E0D 283F      JR      Z,4E4EH      Reprocess command line buffer
4E0F FE40      CP      40H          Test for option 40
4E11 CA564F    JP      Z,4F56H      Copy and Edit file name
4E14 FE50      CP      50H          Test for option 50
4E16 CAC04F    JP      Z,4FC0H      Add suffix to file name
4E19 FE60      CP      60H          Test for option 60
4E1B CA7D50    JP      Z,507DH      Parse command list
4E1E C9        RET                  Unknown option. Return to caller
```

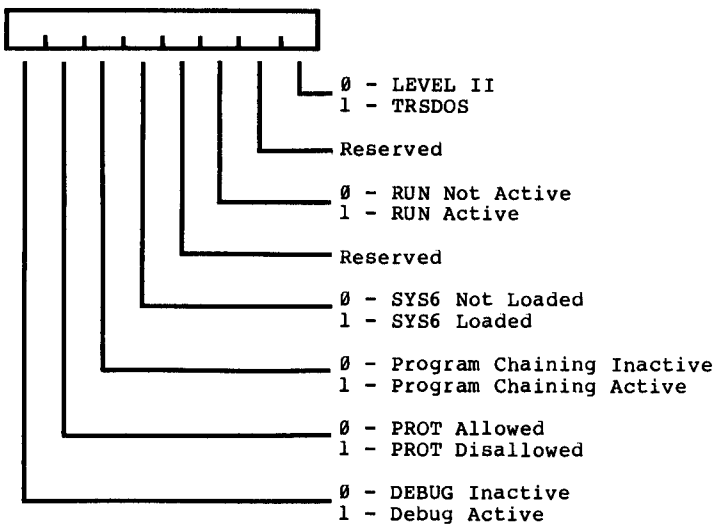
4.1 Request Code 10, 20, and 30

The processing for options 10 and 20 begins at **4E1F**. Processing begins by enabling interrupts and intializing the stack pointer to an empty stack (**41FE**). The latter is an important point. It means that the stack cannot run away because of a bug in an overlay or utility which is executed repeatedly, and it also means that parameters cannot be passed betweeen overlays via the stack.

The instructions at **4E23** and **4E26** are dead; they serve no purpose.

Starting at **4E28** is the first of several tests of the system conditions flags. The byte containing those flags has the following format:

Figure 4.4 *System Condition Flags*



The first test is made to determine if a DEBUG command has been processed previously (Bit 7 of **430F** non-zero, DEBUG is active). If so, the DEBUG flag at **4315** will be set non-zero. This will cause DEBUG to be loaded if a BREAK key is detected during interrupt processing. This code is necessary because prior to entering an overlay, the loader clears the DEBUG flag at **4315**. Thus, DEBUG will not be loaded when executing SYS1/SYS - SYS6/SYS.

If the DEBUG bit in **430F** is not set, or, following the processing if it is set, bit 5 of the system condition flag is tested. This test is also unnecessary as bit 5 is never set, so the jump to **4E40** is always taken. The instructions at **4E38 - 4E3D** are dead.

Beginning at **4E40**, the message 'DOS READY' is displayed, followed by a keyboard read for the next command line. The keyboard read specifies the command line buffer at **4318** as the input buffer, and a maximum of 63 characters for the command length. Shown below is the code used for these operations.

Figure 4.5 *SYS1 Code From 4E1F to 4E4B*

```
*
*       --- Begin processing for options 10 and 20 ---
*
4E1F FB          EI                    Allow interrupts
4E20 31FC41      LD       SP,41FCH     Reset stack pointer
4E23 215943      LD       HL,4359H     This  instruction not used
4E26 CB6E        BIT      05H,(HL)     This instruction not used
4E28 2A0F43      LD       HL,(430FH)   System condition flag
4E2B CB7E        BIT      07H,(HL)     Is DEBUG active
4E2D 2805        JR       Z,4E34H      Jump if no
4E2F 3EC3        LD       A,0C3H       Else, construct a
4E31 321543      LD       (4315H),A    JP XXXX instruction at 4315
4E34 CB6E        BIT      05H,(HL)     Test for program chaining
4E36 2808        JR       Z,4E40H      Jump if no
4E38 3A4038      LD       A,(3840H)    Get keyboard row - 0
4E3B E604        AND      04H          Test for break
4E3D C23040      JP       NZ,4030H     Go recall SYS1 with option 2 if so
4E40 21B851      LD       HL,51B8H     Address of DOS READY message
4E43 CD6744      CALL     4467H        Write DOS READY to video
4E46 211843      LD       HL,4318H     Address of command line buffer
4E49 063F        LD       B,3FH        Max no. of key strokes allowed
4E4B CD4000      CALL     0040H        Get next command from keyboard
```

Following the keyboard read, control returns to **4E4E**. This is also the entry point for a SYS1 call with request option 30. When control passes to this address, the command is assumed to exist in the command line buffer at **4318**.

Command line intrepretation begins at **4E4E**. It consists of four steps. First, the command line is copied to the system DCB at **4480**. Second, the command is compared to a list of TRSDOS commands embedded in SYS1. If there is no match, the third step is taken: the command is compared to a list of TRSDOS commands embedded in SYS6). If this test fails, the fourth step is taken: a CMD suffix is added to the command line in the system DCB, and a subroutine in the nucleus is called to load and execute the file.

If the command line matches any of the SYS1 or SYS6 commands, it is executed directly. SYS1 commands are executed by jumping to the code for the requested operation. Commands in SYS6 are executed after SYS6 has been loaded. A command index is passed to SYS6 in the C register indicating which operation is to be performed.

At **4E4E** a return address of **402D** is PUSH'ed onto the stack. If **4E4E** was entered from the code above, as opposed to a request code 30 call, **402D** will be the only address on the stack, because the stack pointer was initialized at **4E20**. If **4E4E** was entered because of a request code 30, the return address on the stack for the SYS1 call will be preserved; however, this does not mean that control can be returned to that point, as **402D** (which will be POP'ped first) will cause SYS1 to

be called with a request code 10, which, as was mentioned earlier, rewinds the system stack, causing any residue to be lost.

Continuing on, at **4E52** preparations are made to call a subroutine at **4F56** to copy the command line to the system DCB. This is done in anticipation that the incoming command will be a file load (any command not embedded in SYS1 or SYS6).

4F56 is entered with two parameters. The DE register contains the DCB address, and the HL register contains the address of the command line buffer. **4F56** begins by saving the DCB address (DE register), so it can be restored upon exit. It then calls another subroutine at **4FF4** to copy the command to the system DCB.

The subroutine at **4FF4** terminates when the specified number of characters have been copied (eight in this case), or when an illegal character is detected. When control returns to **4F5C**, the character that the copy terminated on is tested to determine the next action to be taken. If, for example, the file name is followed by a '/,' an extension is assumed, and it too is copied. If it is followed by a '(.),' a password is assumed, and it also is copied.

Proceeding down through **4F56**, **4F78** is eventually reached, where a test for a drive specification is made. If one is present, it will also be copied. Finally, the end of the command that has been copied to the DCB is marked with an **03** at **4F86/4F88**.

The instructions at **4F89/4F8A** are unnecessary. At **4F8B/4F8C**, the beginning address of the DCB is restored to the DE register set. At **4F8D** and **4F90**, preparation for (and a call to) an interesting subroutine at **502A** is made.

Although the subroutine at **502A** will be described here, the call to it at **4F90** when **4F56** has been called during SYS1 processing is pointless, and in fact, can cause mysterious problems if an attempt to execute a file named "TO", "ON", or "OVER" is attempted. Files with these names can be created, but not executed because of a bug.

By the time **4F40** is reached, the command line buffer has been copied to the system DCB. The normal thing to do, at this point, would be to return to the caller of **4F56** (in this case, the call came from **4E55**), but instead, the subroutine at **502A** is called to compare the command to the three text strings: TO, ON, and OVER. This call is made because **4F56** can be entered directly in response to a request code 40 which tests for parameter seperators. SYS6 makes such a call to test for delimiters between file names in a copy statement. Unfortunately, if the command (now copied to the DCB), matches one of those text strings, the remainder of the command line buffer is copied into the system DCB, destroying the previous contents (the command line). Below is the code used for the subroutines **4F56** and **4FF4**.

Figure 4.6 *SYS1 Code From 4F56 to 5029*

```
*
* Begin processing for option 50. Copy string specified in
* HL to addr. specified in DE. Source string is assumed to
* be a file name. Dest. addr. is assumed to be a DCB. Source
* string is edited according to file name conventions e.g. no
* special characters etc. On exit, status register is zero if
* no errors were detected, otherwise it is non-zero.
*
* HL= String address
* B = No. chars. in string
* DE= DCB Address
*
4F56 D5          PUSH    DE          Save DCB address.
4F57 0608        LD      B,08H       No. of chars. to examine in string
4F59 CDF44F      CALL    4FF4H       Look for /,; or . in cmd string
4F5C 203A        JR      NZ,4F98H    Jump if no chars. copied to DCB
4F5E FE2F        CP      2FH         Was it terminated by an extension
4F60 2009        JR      NZ,4F6BH    Jmp, no extension specified
4F62 12          LD      (DE),A      Move / to DCB
4F63 13          INC     DE          Bump to next location in DCB
4F64 0603        LD      B,03H       No. of characters to move
4F66 CDF44F      CALL    4FF4H       Move extension to DCB
4F69 203A        JR      NZ,4FA5H    Jump if no extension
4F6B FE2E        CP      2EH         Was there a password present
4F6D 2009        JR      NZ,4F78H    Jmp, no password
4F6F 12          LD      (DE),A      Move . to DCB
4F70 13          INC     DE          Bump to next position in DCB
4F71 0608        LD      B,08H       B = No. of Characters in password
4F73 CDF44F      CALL    4FF4H       Move password to DCB
4F76 202D        JR      NZ,4FA5H    Jump if no password present
4F78 FE3A        CP      3AH         Was there a drive specification (:)
4F7A 2009        JR      NZ,4F85H    No Jump if no,
4F7C 12          LD      (DE),A      Yes. Move : to DCB
4F7D 13          INC     DE          Bump to next position
4F7E 0601        LD      B,01H       Max. number of chars. in drive spec.
4F80 CDF44F      CALL    4FF4H       Copy drive specification to DCB
4F83 2020        JR      NZ,4FA5H    Go if no drive number following :
4F85 4F          LD      C,A         Save terminating characters
4F86 3E03        LD      A,03H       Terminating character for DCB list
4F88 12          LD      (DE),A      To DCB
4F89 AF          XOR     A           Clear status flags
4F8A 79          LD      A,C         Restore term. char. to A
4F8B D1          POP     DE          Restore orgin. of DCB to DE
4F8C D5          PUSH    DE          And save it on stack
4F8D 01A74F      LD      BC,4FA7H    Addr. of TO, ON, OVER phrases
4F90 CD2A50      CALL    502AH       Look for match
4F93 D1          POP     DE          Restore DCB address
4F94 28C0        JR      Z,4F56H     Jump if TO/ON/OVER phrase found
4F96 AF          XOR     A           Signal good status and
4F97 C9          RET                 Return to caller
```

Listing Continued . . .

. . . Continued Listing

```
4FF4    LD    A,B          No. of chars to examine
4FF5    LD    (5026H),A    Save No. of chars. to test
4FF8    INC   B            Adjust for DEC at 5013
4FF9    LD    A,(HL)       Get next symbol from input buff.
4FFA    CP    03H          Was it a terminating character
4FFC    JR    Z,5021H      Jump if yes
4FFE    CP    0DH          Test for Cr
5000    JR    Z,5021H      Jump if Cr
5002    INC   HL           Bump to next char.
5003    CP    30H          Was previous char. a spec. char.
5005    JR    C,5021H      Jump, if yes
5007    CP    3AH          Test for alphanumeric or special char
5009    JR    C,5013H      Jump if numeric
500B    CP    41H          Test for alpha. or spec. symbol
500D    JR    C,5021H      Jump if spec. symbol
500F    CP    5BH          Test for upper case
5011    JR    NC,5021H     Jump if upper case
5013    DEC   B            Count one legit char found
5014    JR    Z,501EH      Jump if all tested
5016    LD    (DE),A       Move to DCB in case it's a file name
5017    XOR   A            Signal one char. found
5018    LD    (5026H),A    Signal one char. copied
501B    INC   DE           Bump to next addr. in DCB
501C    JR    4FF9H        Loop until end of input or
                           specified # of char. copied
501E    INC   B            Count 1 char processed
501F    JR    4FF9H        Continue scan until char.found
5021    LD    C,A          Save last char.
5022    LD    A,03H        Load list terminator
5024    LD    (DE),A       End command with a 03
5025    LD    A,00H        Count of characters tested
5027    OR    A            Set status flags for count
5028    LD    A,C          Last character to A reg.
5029    RET                Return to caller
```

Normally, the subroutine at **502A** is called to match the command in the system DCB against a list of SYS1, or SYS6 commands. The calling sequence is: DE - starting DCB address; BC - starting address of command list to search. The command list has the following format:

Figure 4.7 *Command List Format*

```
DEFM        'TEXTØ1'        A six character keyword
DEFW        ADDR 1          Address to be returned in
                            DE register
DEFM        'TEXTØ2'
DEFW        ADDR 2

  .
  .
  .

DEFB        ØØ              List terminator
```

On exit, HL contains its original value; the AF registers contain a zero or one depending on whether there was a match, or no match respectively; the C register contains the number of keywords compared; and if there was a match, DE contains the routine address following the keyword.

The match subroutine can be divided into three sections. The first (**502A - 5041**) looks for a match between the first character of the command and a keyword. The second (**5042 - 5063**) is called if the first one finds a match. It scans the remainder of the command looking for an exact match with the keyword. The third (**5064 - 507B**) is called by the second when the end of the command is reached. Commands may be terminated by a **03**, or **0D** (carriage return). When a terminator is detected in the command line, the corresponding character in the keyword must be tested to insure it is a blank, otherwise a mis-match has occurred.

Beginning at **502A**, the HL register is saved, as it will be replaced by the BC register (command list address). Next, the count of keywords compared is initialized to one, and the first letter of a keyword is compared to the first letter of the command. If there is a match, control goes to section two (**5042**).

If there is no match, the HL register pair is bumped by eight to skip to the next keyword in the list, the count of keywords compared is incremented, and a test for the end of the command list is performed. If the end of the list has not been reached, control goes to **502F**, where the first letter of the next keyword is compared. The basic loop formed goes from **502F** to **503C**. If the end of the command list is reached with no match, the HL register is restored, and control returns to caller with a non-zero status. Following is the code used for this section:

Figure 4.8 *SYS1 Code From 502A to 5041*

```
*
* Compare list pointed to by BC against list pointed to by DE
*
502A E5          PUSH    HL              Save input string addr.
502B 60          LD      H,B             HL = Addr. of command
502C 69          LD      L,C             Tables to search
502D 0E01        LD      C,01H           C = Number of lists searched
502F 1A          LD      A,(DE)          .   Fetch a char.from search list
5030 BE          CP      (HL)            .   Compare it to input string
5031 280F        JR      Z,5042H         .   Jump if first char. matches
5033 C5          PUSH    BC              .   .   No match. Go to next list
                                         .   .   Save  caller's BC
5034 010800      LD      BC,0008H        .   .   Add 8 to current list to
5037 09          ADD     HL,BC           .   .   get addr. of next list
5038 C1          POP     BC              .   .   Restore caller's BC
5039 0C          INC     C               .   .   Bump count of lists searched
503A 7E          LD      A,(HL)          .   .   Fetch 1st char. of next list
503B B7          OR      A               .   .   test for end of list
503C 20F1        JR      NZ,502FH        .   .   Loop, not end of list
503E E1          POP     HL              .   .   Restore caller's HL
503F F601        OR      01H             .   Signal no match
5041 C9          RET                     .   Return to caller
```

Section two is entered when a match has been found between the first letter of a keyword and the command. The B register is initialized to 5 (the number of remaining letters to be matched); the HL register (keyword list address) is saved, so it can be restored in case of a mis-match; and the same is done for the DE register (command list address).

Next, the next letter from the command list is fetched and compared to an end of command terminator, (**03** or **0D**). If a terminator is detected, control goes to section three. Otherwise, the letter is compared to the next letter in the keyword, and, if equal, the B register is decremented and tested for zero. If the B register is non-zero, control returns to **5046** (the start of the loop) where the next letter is fetched and tested.

When the B register becomes zero, control falls through to **5056**. Here the starting address plus one of the keyword addresses is restored, so the end of the keyword can be computed. This is necessary because the length of the particular command is not known. But since the maximum length of all keywords is known, the length of any particular command can be computed.

After locating the end of the keyword, the address of the code associated with the command is loaded into the DE. The DCB address is restored to the HL; a good status value is loaded into the A register, and control returns to the caller. Below is the code used for section two.

Figure 4.9 *SYS1 Code From 5042 to 5063*

```
*
* First char. in list matches. Compare remainder of list.
*
5042 0605       LD      B,05H         .  .   No. of chars. left to match
5044 E5         PUSH    HL            .  .   Save input string addr.
5045 D5         PUSH    DE            .  .   Save search list addr. Save
                                      .  .   starting addr. of command string
5046 13         INC     DE            .  Bump to next char. of input list
5047 23         INC     HL            .  and companion list
5048 1A         LD      A,(DE)        .  Fetch a char. from input list
5049 FE03       CP      03H           .  Test for end of string
504B 2827       JR      Z,5074H       .  Go if end of input string
504D FE0D       CP      0DH           .  Else test for end of line
504F 2823       JR      Z,5074H       .  Jump if end of input string
5051 BE         CP      (HL)          .  Does char. match search list
5052 2010       JR      NZ,5064H      .  No, go test for ending byte
5054 10F0       DJNZ    5046H         .  Yes, loop to test next char.
                                      .  until all chars. tested
5056 D1         POP     DE            DE = 2nd char. of matching
                                      search list
5057 79         LD      A,C           A = No. of lists searched
5058 C1         POP     BC            Starting addr. of matching cmd
5059 210600     LD      HL,0006H      HL = Length of command
505C 09         ADD     HL,BC         HL = Ending addr. of command
505D 4F         LD      C,A           C = List number
505E 5E         LD      E,(HL)        E = LSB of command addr.
505F 23         INC     HL            Bump to MSB
5060 56         LD      D,(HL)        D = MSB of command addr.
5061 E1         POP     HL            HL = Starting addr. of
                                      input string
5062 AF         XOR     A             Signal good status
5063 C9         RET                   Return to caller
```

The third section is entered when a terminating character is found in the command, or there is a mis-match between the keyword and the command.

In the terminating character case, the next letter from the keyword is examined. If it is a blank, a match has been found, and control goes to **5056**, where the proper exit is made (see above for a description of the code at **5056**).

In the mis-match case, the current letter in the command is tested to determine if it is blank, '/,' '.,' ':,' '=,' or other special character. If so, control goes to **5074**, where the test for end of keyword is made. Basically, these special characters are treated as terminators.

If the current letter in the keyword is not a blank, the end of the keyword has not been reached, and control passes to **5079**, where the DE register is restored to the beginning address of the command and the HL is set to the starting address of the keyword. Control then passes to **5034**, where the HL register is bumped to the next keyword and testing resumes. This code covers the case where two commands have similar, but not identical, names. Following is the code used in section three:

Figure 4.10 *SYS1 Code From 5064 to 507B*

```
*
*
*
5064 FE30       CP      30H             .  .   Was it a special char.
                                        .  .   (e.g. / . - + - *)
5066 380C       JR      C,5074H         .  .   Jump if yes
5068 FE3A       CP      3AH             .  .   No, was it a digit 0 - 9
506A 380D       JR      C,5079H         .  .   Jump if yes
506C FE41       CP      41H             .  .   No, was it a special char.
                                        .  .   (e.g : ; < = > ? @)
506E 3804       JR      C,5074H         .  .   Jump if yes
5070 FE5B       CP      5BH             .  .   No, was it alpha A - Z
5072 3805       JR      C,5079H         .  .   Jump if yes
5074 7E         LD      A,(HL)          .  .   No, fetch current char.
5075 FE20       CP      20H             .  .   Is it a space
5077 28DD       JR      Z,5056H         .  .   Jump, if yes
5079 D1         POP     DE              .      No, clear stack
507A E1         POP     HL              .      Restore input string addr.
507B 18B6       JR      5033H           Get next list addr. and retest
```

When control returns to **4F93**, the system DCB address is restored and the status of the A register is tested. If it contains a zero, there was a match between one of the keywords in the list beginning at **4FA7** and the command. In that case, the remainder of the command line buffer is copied to the DCB. As noted before, this is an error, and this path is taken only when executing a command that matches one of the keywords in the list at **4FA7**.

A more likely exit, and the only correct one, is at **4F96** and **4F97**, where a good status is signalled, and control returns to the caller (in this case **4E58**).

Returning to **4E58** (request code 10, 20, and 30 processing), the status register is tested. If non-zero, an illegal character was detected in the command string. Control will pass to **4E87**, where the message "WHAT" will be displayed and SYS1 will be re-entered with a request code 20 (via **4030**).

Assuming no illegal character terminated the command, a test for an '*' as the first character is made. This test is necessary because the subroutine at **4F56** will allow a special name consisting of an '*' followed by two alphanumeric characters to be copied into the DCB. If the file name begins with an '*,' the "WHAT" prompt is displayed and SYS1 is re-entered via a call to **4030**.

If there were no illegal characters in the command, it will be compared to a list of TRDOS commands. The subroutine at **502A** (described earlier) will be used for the comparison.

The first test for a TRSDOS command is at **4E65**. The test is made by calling a subroutine that compares a list of keywords against a fixed list (the command line). The keywords for this test are the embedded SYS1 commands. The subroutine used for the comparison is located at **502A**.

Prior to calling **502A**, the HL register is saved, and the BC register is loaded with the address of internal SYS1 commands. These commands are processed entirely by SYS1. A description of this list follows.

Figure 4.11 *SYS1 Internal Command Addresses*

ADDRESS OF KEYWORD	KEYWORD	ADDRESS OF ACTION ROUTINE
4EA4	'BASIC2'	514C
4EAC	'DEBUG '	5162
4EB4	'TRACE '	5191

When control returns, a test for a match between the command and any entry in the list is made. If there is a match, control is passed to **4E84**.

At **4E84**, the current command line buffer address is restored to HL (this is really unnecessary since it was preserved by the subroutine call), and the address of the routine to be executed is PUSH'ed onto the stack (simulating a CALL instruction). Following that, a RET is executed, which passes control to the address associated with the keyword.

If the command did not match any of the entries in the list at **4EA4** (SYS1 commands), another call to **502A** is made with a list of TRSDOS commands processed by SYS6. This list begins at **4EBD**; it is described below. Again, upon return, a test for a match with any entry in the list is made, and, if true, control goes to **4E84**.

If the command does not match any of the SYS1 or TRSDOS commands, it is assumed to be a filename. At **4E72**, bit 4 of the system condition flag is cleared, indicating a user file load. Next, at **4E7A** a call is made to a subroutine at **4FC0** to add the CMD suffix to the filename in the DCB. At this point, the DE register contains the address of the system DCB holding the name of the file to be loaded. Control is now passed to the system file loader in the nucleus to load and execute the file. This is done by the jump at **4E81** to **4433**. **4433** contains a second jump to **4C06**. At **4C06**, the file will be OPEN'ed, and its permission flags tested before the file is loaded and executed. Following is the code used for these operations:

Figure 4.12 *TRSDOS Command List*

ADDRESS OF KEYWORD	KEYWORD	COMMAND INDEX
4EBD	'APPEND'	Ø1
4EC4	'ATTRIB'	Ø2
4ECD	'AUTO '	Ø3
4ED5	'CLOCK '	Ø4
4EDE	'COPY '	Ø5
4EE4	'DATE '	Ø6
4EED	'DEVICE'	Ø7
4EF5	'DIR '	Ø8
4EFD	'DUMP '	Ø9
4FØ5	'FREE '	1Ø
4FØD	'KILL '	11
4F15	'LIB '	12
4F1D	'LIST '	13
4F25	'LOAD '	14
4F2D	'PRINT '	15
4F35	'PROT '	16
4F3D	'RENAME'	17
4F45	'TIME '	18
4F4D	'VERIFY'	19

Figure 4.13 *SYS1 Code From 4E4E to 4EA3*

```
4E4E Ø12D4Ø     LD      BC,402DH    Return point in SYSØ
4E51 C5         PUSH    BC          To stack
4E52 118Ø44     LD      DE,4480H    Address of system DCB
4E55 CD564F     CALL    4F56H       Move file name to DCB (command line)
4E58 C2874E     JP      NZ,4E87H    Go if illegal char. found
4E5B 1A         LD      A,(DE)      Fetch first character of command
4E5C FE2A       CP      2AH         Test for *
4E5E CA874E     JP      Z,4E87H     Go if command starts with * (error)
4E61 E5         PUSH    HL          Save end position in command list
4E62 Ø1A44E     LD      BC,4EA4H    Address of SYS1 commands
```

Listing Continued . . .

. . Continued Listing

```
4E65 CD2A50      CALL    502AH          Test for SYS1 cmd
                                        BASIC2/DEBUG/TRACE
4E68 281A        JR      Z,4E84H        Jump if SYS1 command. Execute CMD
4E6A 01BD4E      LD      BC,4EBDH       Else load addr. of other DOS commands
4E6D CD2A50      CALL    502AH          Compare current command against
                                        list of SYS6 commands
4E70 2812        JR      Z,4E84H        Jump if a match found
4E72 3A0F43      LD      A,(430FH)      Fetch system condition flag
4E75 CBA7        RES     04H,A          Clear SYS6 loaded flag
*
*         Command is neither SYS 1 nor SYS 6 command.
*       Add CMD suffix to command and load file from disk.
*
4E77 320F43      LD      (430FH),A      Restore updated condition flag
4E7A 21B551      LD      HL,51B5H       Address of CMD message
4E7D CDC04F      CALL    4FC0H          Add CMD suffix to file name
4E80 E1          POP     HL             Clear DCB address from stack
4E81 C33344      JP      4433H          Enter routine to load and execute file
4E84 E1          POP     HL             Clear stack
4E85 D5          PUSH    DE             SYS1 addr for DOS cmd to stack
4E86 C9          RET                    Go to SYS1 addr for DOS command.
                                        (514C, 5162, 5191, or 4E90)
*
*
*
4E87 21C351      LD      HL,51C3H       Address of "WHAT" message
4E8A CD6744      CALL    4467H          Print "WHAT"
4E8D C33040      JP      4030H          Go wait for next command
4E90 3A0F43      LD      A,(430FH)      Get system condition flags
4E93 CB67        BIT     04H,A          Test if SYS6 currently loaded
4E95 C20052      JP      NZ,5200H       Jump if yes
4E98 3A0F43      LD      A,(430FH)      Else reload system condition flag
4E9B CBE7        SET     04H,A          And clear SYS6 loaded flag
4E9D 320F43      LD      (430FH),A      Restore system condition flags
4EA0 3E88        LD      A,88H          DOS overlay req. for SYS6 option 0
4EA2 F5          PUSH    AF             Dummy push for overlay loader
4EA3 EF          RST     28H            Load and execute SYS6
```

4.2 Request Code 40.

This entry point begins at **4F56**. It is used to edit a command line while moving it to a DCB. The address of the command line is given in the HL register, and the DCB address is given in the DE registeer. A complete description of this function may be found in Section 4.1

4.3 Request Code 50.

This entry point begins at **4FC0**. It is used to add a suffix to a file name. The code is generalized to the point where the file name may already contain a suffix, a password, or a drive specification.

The entry conditions are: DE contains the address of the DCB. HL contains the address of a three letter suffix. The code begins by saving the DE and HL, and then exchanging them.

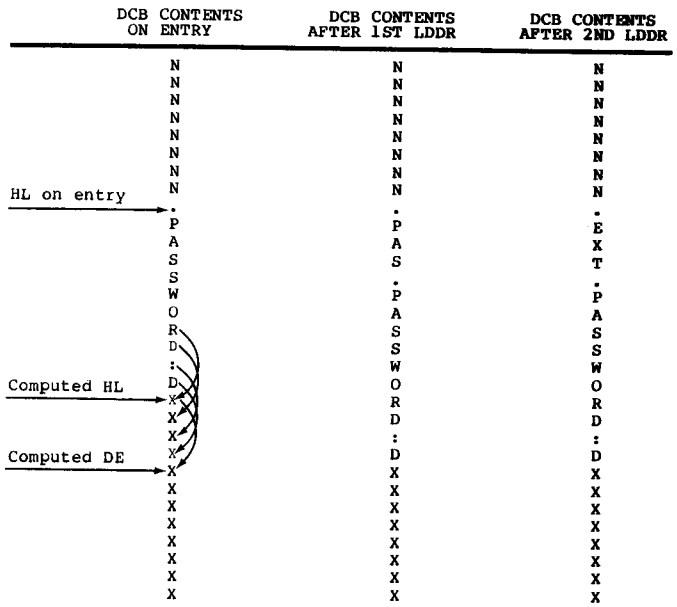
At **4FC4** preparations are made to scan the 9 characters of the file name for a '/,' ':,' '.,' or end of name (**03** or **0D**). The scan loop starts at **4FC6** and extends through **4FD6**. If a '/' is detected (test at **4FC7 - 4FC9**), control returns to the caller without adding the suffix, since the file already has one.

Following the test for a '/,' is an indirect test for a '.'(password) followed by aother indirect test for a ':.' If either condition is true, or if the end of the name is reached (usually terminated by an **03**) control goes to **4FDB** where the suffix is added to the file name. If a file name of 9 characters is found, control returns to the caller without adding the suffix.

The code from **4FDB** through **4FF3** adds the suffix to the file name. This code expects the HL register to point to the position in the name where the suffix is to be added. Because a password, or a drive specification may follow the file name, all characters following the name must be moved down to create a space for the suffix.

This code can best be understood with the aid of the following diagram.

Figure 4.14 *DCB Positioning for Adding Suffix*



On entry, HL points to the exact position where the suffix is to be added. From that point on the remainder of the file name must be moved down 4 places. At **4DFE**, a new value for the HL register is computed. Since a LDDR instruction will be used to move the remainder of the file name down, the ending address for the existing name must be computed. Because the exact length of the rest of the name is not known, a maxiumum length of 15 is assumed. This value is added to the HL register at **4FDE**, giving the computed HL shown in the illustration above.

The computed HL is then copied to the DE register and incremented by four. This gives the destination address for the block move. It is four bytes beyond the end of the file name. Next, the C register is incremented to give the number of bytes to move (16 in this case), and the block move instruction is executed giving the 'after 1st LDDR' picture shown above.

Next, the starting address of the suffix is loaded and the HL register set. It must be incremented twice to point to the end of the suffix, because an LDDR instruction will be used to copy it to the DCB. The LDDR instruction at **4FED** copies the suffix into the DCB. This instruction is used because the DE register is already pointing to the ending character position for the suffix in the DCB. After the LDDR instruction has been executed, the DCB appears as shown in the 'after 2nd LDDR' in the illustration above. Finally, a '/' is inserted between the first part of the file name and the suffix, the DE is restored to the DCB address and control returns to the caller. Below is the code used for this processing:

Figure 4.15 *SYS1 Code From 4FC0 to 4FF3*

```
*
* Begin processing for option 50. Add /CMD suffix to file
* name in DCB. Skip to end of file name (find last char of
* file name before drive specification). Move all chars after
* file name down 4 positions (create a hole for /CMD suffix).
* Add CMD suffix, if a / is detected following file name do
* not add suffix
*
4FC0 D5          PUSH    DE          Save DCB addr.
4FC1 E5          PUSH    HL          Save appendage list
4FC2 EB          EX      DE,HL       DE = appendage list. HL = DCB addr
4FC3 23          INC     HL          Point to the 2nd char. in DCB
4FC4 0609        LD      B,09H       Max.# of char.to test.
4FC6 7E          LD      A,(HL)      .   Get a char.from DCB
4FC7 FE2F        CP      2FH         .   Test for / (extension)
4FC9 280D        JR      Z,4FD8H     .   Go if / found (don't add suffix)
4FCB 380E        JR      C,4FDBH     .   Or go if special char.
                                     .   found (do not add suffix)
4FCD FE3A        CP      3AH         .   Test for : (drive spec.)
4FCF 3804        JR      C,4FD5H     .   Jump if numeric found
4FD1 FE41        CP      41H         .   Test for letter
4FD3 3806        JR      C,4FDBH     .   Jump if not letter. End of
                                     .   file name in DCB (03) assumed
4FD5 23          INC     HL          .   We have a letter. Bump to next
4FD6 10EE        DJNZ    4FC6H       .   char. & loop till all examined
```

Listing Continued . . .

. . . Continued Listing

```
4FD8 E1          POP     HL          Restore appendage list addr.
4FD9 D1          POP     DE          Restore DCB addr.
4FDA C9          RET                 Return to caller w/o adding suffix
*
*
*
4FDB 010F00      LD      BC,000FH    BC = 15, max distance to end of DCB
4FDE 09          ADD     HL,BC       Compute addr. of last byte used
4FDF 54          LD      D,H         DE = last byte
4FE0 5D          LD      E,L         used in DCB
4FE1 13          INC     DE          Now, add 4 to last used
4FE2 13          INC     DE          addr. so we can move
4FE3 13          INC     DE          file name down to create
4FE4 13          INC     DE          A "hole" for the /CMD suffix
4FE5 03          INC     BC          BC = number of bytes to move down
4FE6 EDB8        LDDR                Move data after file name down 4
4FE8 E1          POP     HL          HL = start addr. of CMD text
4FE9 23          INC     HL          Skip to
4FEA 23          INC     HL          end of list for LDDR instr.
4FEB 0E03        LD      C,03H       C = number of chars. to copy
4FED EDB8        LDDR                Copy CMD text to DCB
4FEF 3E2F        LD      A,2FH       A = "1"
4FF1 12          LD      (DE),A      Separate file name & suffix with "/"
4FF2 D1          POP     DE          Restore start of DCB addr.
4FF3 C9          RET                 Return to caller
```

4.4 Request Code 60

This entry point begins at **507D** and continues through **514A**. It is called from places outside of SYS1 (usually SYS6) to parse parameters enclosed in parenthesis.

On entry, HL points to a parameter list that must start with a space on a '(.' DE points to a command list whose format is indentical to that used in SYS1. Fields within the parentheses may be seperated by a comma or equal signs. If a comma is found at the end of field, another field is presumed to follow, and it will be processed as well. Equals are assumed to precede a numeric value which is to be converted to binary. Hexadecimal values must be preceded with an 'X,' otherwise, a decimal field is assumed.

Control returns to the caller when the closing parenthesis is found, or if a left parenthesis is not found at the start of the command string. The A register will contain a zero if a closing parenthesis was found. Any error will force control back to the caller with a non-zero status.

A match between the interior of the parenthesis and the command list is signalled to the caller by returning a **FFFF** at the address specified after the keyword. If the matching field is followed by an equals sign, the numeric value following the equals sign will be returned instead of an **FFFF**. Note, this is a slightly different use of the address following keywords as compared to command lists embedded in SYS1.

Starting at **507D** is a series of tests for a carriage return, right parenthesis, or a space. Spaces are ignored, and if found, the HL is advanced to the next character in the command string and the tests are re-executed. A carriage return will terminate the subroutine.

When a right parenthesis is found, control goes to **508E**, where the caller's DE is saved and preparations are made to call **4FF2** to copy the next six characters into an intermediate buffer at **51AF**. At **5091**, **4FF2** is called to copy the next field to the intermediate buffer. Upon return, the command string pointer is backspaced to point to the terminating character and the caller's command list address is restored to the DE register set.

If no characters were copied, an error has occurred, and control is returned to the caller with a non-zero status. Otherwise, the caller's command list address is saved on the stack once again, and copied into the BC register set.

Next, at **509E**, the intermediate buffer address (**51AF**) is loaded into the DE register, and the subroutine at **50A2** is called to determine if there is a match between the buffer and the caller's command list. If there is a match, the DE register is restored at **50A6** and control returns to the caller with a non-zero status.

If a match did occurr, the terminating character is compared to an equals sign (this happens at **50A9**). If the terminating character is an equals sign, control goes to **50C1**, where a numeric field is processed.

If the field was not terminated with an equals sign, an **FFFF** is stored at the address carried in the command list. This address will be left in the DE register by the subroutine at **50A2** if there is a match. The code from **50AE** through **50B4** stores a **FFFF** at the address specified in DE.

Following that, the terminating character is retested for equaling a comma. If it is a comma, control goes to **508E**, where the next field is extracted and processed.

Thus, multiple fields within a set of parentheses can be processed. Note that the origin of the caller's command list address is preserved at **508E** and restored at **50B5**; thus the fields can be in any order.

If the termrinating character was not a comma, it is compared to a right parenthesis. If it is a '),' the HL register is incremeted to the first character beyond the terminal character, the A register status is set to zero (good status), and control returns to the caller. If it does not match, control is returned to the caller with a non-zero status.

Quantitities following an equals sign are processed at **50C1**. There are three possibilities for these quantities. Hexadecimal values are one; they must be

preceded by an 'X.' Decimal values are another, they must begin with a digit in the range 0 - 9; the third possibility is the quantity YES, NO, or OFF.

Beginning at **50C1**, the first character following the equals is tested. If it is an 'X' (hex value follows), control is passed to **50D2**. If it is a digit 0 - 9 (decimal value follows), control goes to **50DA**. If it is neither, a subroutine at **50DF** is called to test for a 'Y,' 'N,' or 'OF.' This subroutine returns the following values in the BC register depending on the next character in the command: Y, BC = **FFFF**; N, BC = **0000**; OF, BC = **0000**; no match, BC = **0000**.

After a 'Y,' 'N,' or 'OF' has been located, a loop from **50FA** to **5102** will scan the command list until a right parenthesis or comma is detected. At that point, control will be returned to the caller.

The code at **50D2** calls a subroutine at **511F** to convert an ASCII string to a hex value. When control returns from **511F**, an error check is performed. If there was no error, control goes to **50B0**, where the hex value is saved and the rest of the command string is processed. If an error occurred, control is returned to the caller with a bad status (non-zero) via the return sequence at **50A6**.

The code at **50DA** is executed if a decimal value follows an equals sign. It is functionally equivalent to the code at **50D2**, except that it calls a subroutine at **5104** to convert ASCII to decimal.

The ASCII-to-decimal and ASCII-to-hexadecimal subroutine at **5104** and **511F** will not be discussed in detail. Both expect the HL register to point to the ASCII string to be converted, and both leave the binary equivalent of the value in the DE register. Following is the code used for request code 60:

Figure 4.16 *SYS1 Code From 507D to 514A*

```
*
*         ---  Begin processing for option 60 parse  ---
*
507D 7E         LD      A,(HL)          .   Get a character
507E FE0D       CP      0DH             .   Test for end of line (C.R.)
5080 C8         RET     Z               .   Exit if a "CR" found
5081 FE28       CP      28H             .   Test for left parenthesis
5083 2809       JR      Z,508EH         .   Jump if "("
5085 FE20       CP      20H             .   Test for space
5087 2802       JR      Z,508BH         .   Jump if space
5089 B7         OR      A               .   Set status flags
508A C9         RET                     .   Return to caller
508B 23         INC     HL              .   Bump to next char.
508C 18EF       JR      507DH           .   Go test again
508E D5         PUSH    DE              .   Save caller's cmd list addr
508F 0606       LD      B,06H           .   B = Number of chars. to copy
5091 11AF51     LD      DE,51AFH        .   Buffer addr.
5094 23         INC     HL              .   Bump to first char. after "("
5095 CDF44F     CALL    4FF4H           .   Copy 6 chars or until ")" found
5098 2B         DEC     HL              .   Backspace over terminating 03
```

Listing Continued . . .

. . . Continued Listing

```
5099 D1          POP     DE            .   Restore caller's DE
509A C0          RET     NZ            .   Exit if no chars. copied (error)
509B D5          PUSH    DE            .   Save caller's DE
509C 42          LD      B,D           .   Caller's command list table
509D 4B          LD      C,E           .   Addr. to BC for routine at 502A
509E 11AF51      LD      DE,51AFH      .   Parsed command line
50A1 CD2A50      CALL    502AH         .   Look for cmd in caller's cmd list
50A4 2802        JR      Z,50A8H       .   Jump if a match found
50A6 D1          POP     DE            .   .   Restore caller's DE
50A7 C9          RET                   .   .   Return with error status
*
*
50A8 7E          LD      A,(HL)        .   .   Refetch char. from input string
50A9 FE3D        CP      3DH           .   .   Compare it to an =
50AB 2814        JR      Z,50C1H       .   .   Jump if =
50AD 01FFFF      LD      BC,0FFFFH     .   .   Value returned if no = parameter
50B0 79          LD      A,C           .   .   .   Save LSB of addr.
                                       .   .   .   or hex value
50B1 12          LD      (DE),A        .   .   .   In buffer
50B2 13          INC     DE            .   .   .   Bump to next loc.
50B3 78          LD      A,B           .   .   .   Get MSB of addr.
50B4 12          LD      (DE),A        .   .   .   and save in buffer
50B5 D1          POP     DE            .   .   .   Restore caller's
                                       .   .   .   command list address
50B6 7E          LD      A,(HL)        .   .   .   Refetch Char. from
                                       .   .   .   input string
50B7 FE2C        CP      2CH           .   .   .   Compare with '
50B9 28D3        JR      Z,508EH       .   .   .   Jump if '
50BB FE29        CP      29H           .   .   .   Not ', compare it to a )
50BD C0          RET     NZ            .   .   .   Exit if not a )
50BE 23          INC     HL            .   .   .   Bump to next char.
50BF AF          XOR     A             .   .   .   Signal good status
50C0 C9          RET                   .   .   .   Return to caller
50C1 23          INC     HL            .   .   .   Bump to first char.
                                       .   .   .   after =
50C2 7E          LD      A,(HL)        .   .   .   Fetch char.
50C3 FE58        CP      58H           .   .   .   Test for X (hex flag
50C5 280B        JR      Z,50D2H       .   .   .   Go convert if hex value
50C7 FE41        CP      41H           .   .   .   Test for digit
50C9 380F        JR      C,50DAH       .   .   .   Jump if digit
50CB CDDF50      CALL    50DFH         .   .   .   Alpha, go test for N, Y, OF
50CE 28E0        JR      Z,50B0H       .   .   .   Jump if N, Y, or OF found
50D0 18D4        JR      50A6H         .   .   .   Error, return to caller
50D2 23          INC     HL            .   .   .   Bump to next char.
50D3 CD1F51      CALL    511FH         .   .   .   Convert ASCII hex to binary
50D6 28D8        JR      Z,50B0H       .   .   .   If no error save hex value
50D8 18CC        JR      50A6H         .   .   .   Else, return to caller
50DA CD0451      CALL    5104H         .   .   .   Go get decimal value
50DD 18D1        JR      50B0H         .   .   .   and return to caller
*
*           --- Test for YES, NO or OFF parameter ---
*
50DF 010000      LD      BC,0000H      Signal N,Y,or OF
50E2 7E          LD      A,(HL)        Fetch first symbol past "(" or =
50E3 FE59        CP      59H           Test for Y
```

Listing Continued . . .

. . . Continued Listing

```
50E5 2810       JR      Z,50F7H      Jump if Y
50E7 FE4E       CP      4EH          No, test for N
50E9 280F       JR      Z,50FAH      Jump if N
50EB FE4F       CP      4FH          Not Y/N, test for O
50ED C0         RET     NZ           Return if not O
50EE 23         INC     HL           Char. is O, bump to next char
50EF 7E         LD      A,(HL)       Fetch char. after O
50F0 FE46       CP      46H          And test for F
50F2 2806       JR      Z,50FAH      Jump if OF found
50F4 FE4E       CP      4EH          Not F, test for N
50F6 C0         RET     NZ           Return if ON
50F7 01FFFF     LD      BC,0FFFFH    Signal ON
50FA 23         INC     HL           .   Bump to next input string char.
50FB 7E         LD      A,(HL)       .   Fetch next char.
50FC FE29       CP      29H          .   Test for ")"
50FE C8         RET     Z            .   Return if )
50FF FE2C       CP      2CH          .   Test for ","
5101 C8         RET     Z            .   Return if ,
5102 18F6       JR      50FAH        .   Loop until ) or , found
*
*          ---  ASCII to binary conversion routine  ---
*
5104 010000     LD      BC,0000H     Clear accumulator
5107 7E         LD      A,(HL)       .   Fetch an ASCII character
5108 D630       SUB     30H          .   Get its decimal equivalent
510A D8         RET     C            .   Exit if not a digit
510B FE0A       CP      0AH          .   Test for value greater than 9
510D D0         RET     NC           .   Exit if char. greater than 9
510E E5         PUSH    HL           .   Save input string addr.
510F 60         LD      H,B          .   HL = Accumulated
5110 69         LD      L,C          .   Value, thus far
5111 29         ADD     HL,HL        .   Value, * 2
5112 29         ADD     HL,HL        .   Value, * 4
5113 09         ADD     HL,BC        .   Value, * 5
5114 29         ADD     HL,HL        .   Value, * 10
5115 0600       LD      B,00H        .   Prohibit overflow during add
5117 4F         LD      C,A          .   BC = Current digit
5118 09         ADD     HL,BC        .   Add it to previous value
5119 44         LD      B,H          .   Then, move current
511A 4D         LD      C,L          .   value to BC
511B E1         POP     HL           .   Restore cmd string address
511C 23         INC     HL           .   Bump to next char.
511D 18E8       JR      5107H        .   Loop till end of line detected
*
*     Scan input string following ( or = look for Y, N, or OF. If
*     found, return W/BC = 0 after skipping to next , or ) in cmd
*     string. Look for ON. If found, return W/BC = -1 and
*     position to next , or ) in cmd string. If Y, N, or O does
*     not follow as next char, exit W/BC = O.
*
511F 010000     LD      BC,0000H     Zero accumulator
5122 7E         LD      A,(HL)       Get a char. from cmd string
5123 FE27       CP      27H          Compare it to an ' (apostrophe)
5125 C0         RET     NZ           Exit if val. not preceded by ' (error)
5126 23         INC     HL           Bump to first digit
```

Listing Continued . . .

. . . Continued Listing

```
5127 7E          LD      A,(HL)          .   Fetch a digit
5128 D630        SUB     30H             .   Convert it to binary
512A 380A        JR      C,5036H         .   Jump if end of input string
512C FE0A        CP      0AH             .   Test for digit 0-9
512E 380E        JR      C,513EH         .   Jump if 0-9
5130 D607        SUB     07H             .   Test for lettdr A-F
5132 FE10        CP      10H             .   And adjust binary value
5134 3808        JR      C,513EH         .   Jump if A - F
5136 7E          LD      A,(HL)          .   Refetch non-digit char.
5137 FE27        CP      27H             .   Test for end of hex digit (')
5139 23          INC     HL              .   Bump to next input string char.
513A C8          RET     Z               .   Exit if done
513B 2B          DEC     HL              .   Else back-up to illegal character
513C AF          XOR     A               .   Signal an error
513D C9          RET                     .   and return to caller
*
*       Multiply current digit by 16 and add to accumulator.
*
513E E5          PUSH    HL              .   Save input string addr.
513F 60          LD      H,B             .   Move current hex binary value
5140 69          LD      L,C             .   To HL
5141 29          ADD     HL,HL           .   Multiply current hex value by
5142 29          ADD     HL,HL           .   16 using additions
5143 29          ADD     HL,HL           .   *8
5144 29          ADD     HL,HL           .   *16
5145 44          LD      B,H             .   Save MSB of new hex value
5146 85          ADD     A,L             .   Add current hex digit
5147 0F          LD      C,A             .   BC = Current hex value
5148 E1          POP     HL              .   Restore input string addr.
5549 23          INC     HL              .   Bump to next digit
514A 18DB        JR      5127H           .   Loop till non-hex found
```

notes

5

5.0 SYS2/SYS

SYS2/SYS is part of the TRSDOS file management system. It is called by the nucleus in response to various I/0 calls, and by SYS6. SYS6 uses various internal subroutines in SYS2. These subroutines are called directly from SYS6 after SYS2 has been loaded through an OPEN call. There are three entry points in SYS2; they are:

Figure 5.1 *SYS2 Entry Points*

SYS2 OPTIONS	Function
1Ø	Open a file.
2Ø	INIT a file.
3Ø	Build an overflow directory entry.

SYS2 begins by isolating the options field of the request code and branching to the corresponding entry point. Following is the code used for this operation:

Figure 5.2 *SYS2 Code From 4E00 to 4E1E*

```
4E00 E670       AND     70H             Isolate option
4E02 FE10       CP      10H             Test for OPEN request
4E04 CA124E     JP      Z,4E12H         Jump if OPEN
4E07 FE20       CP      20H             Test for INIT request
4E09 CAD84E     JP      Z,4ED8H         Jump if INIT
4E0C FE30       CP      30H             Test for create overflow entry
4E0E CA504F     JP      Z,4F50H         Jump if create overflow entry
4E11 C9         RET                     None of the above - Ret caller
*
*                       Begin OPEN Processing
*
4E12 CD9648     CALL    4896H           Save reg. setup return sequence
4E15 78         LD      A,B             A = Record length
4E16 32AF4F     LD      (4FAFH),A       Save record length
4E19 22C04F     LD      (4FC0H),HL      Callers sector buffer address
4E1C DDE5       PUSH    IX              DCB addr. to HL via stack
4E1E E1         POP     HL              HL = DCB addr.
```

5.1 Request Code 10 (OPEN)

A request code of 10 selects OPEN processing in SYS2. The procedure used is straightforward. It consists of the following steps:

1. Save the registers and setup the exit sequence.
2. Copy the file name to an internal buffer area.
3. Compute a hash code for the file name.
4. Compute a password value if a password was specified.
5. Read the HIT sectors and look for a matching hash code.
6. Read the directory entry and construct the DCB.

The code for OPEN processing begins at **4E12**. It starts by calling a subroutine at **4896** in the nucleus to save the register context. This subroutine, also, saves a return address of **48B3** on the stack, which restores the context and returns to the caller.

After the registers have been saved, the record length and the buffer address are saved in locations within SYS2 where they will be used to initialize a directory entry and DCB.

At **4E15**, a subroutine at **5027** is called to process the file name. The processing consists of separating the file name into its constituent parts and storing them in separate buffer areas within SYS2. The file name will be copied to an 8-byte buffer at **515D**, the extension will be copied to **5165**, the password will be copied to a buffer at **5155**, and the drive specification, if supplied will be saved at **5154**. If no drive was specified, **5154** will contain a -1 (**FF**). For a detailed description of this subroutine, see Section 5.4.

When control returns from **5027**, a test for errors is performed. If any error occurred (illegal character in file name, illegal drive specification, etc.), control will be returned to the caller with the proper error code in the A register.

Assuming no errors were detected, processing continues at **4E23** and **4E26**, where a subroutine at **509B** is called to compute a hash code for the file name in the buffer beginning at **515D**. The hash code returned will be a number between **01** and **FF**. This code is saved at **4E4E** for comparison with other hash codes when the HIT is read. For a description of the hash subroutines, see Section 5.5.

Continuing at **4E2C** and **4E2F** is a call to **50D1** to compute the password code. The password buffer begins at **5155**. The password code will be returned as a 16-bit value in the HL register. It will be saved at **5168** and **516A** for storage into the directory sector and DCB. Following is the code used to compute a password value:

Figure 5.3 *SYS2 Code From 50D1 to 50FC*

```
*
*       Compute encode of password in buffer at 5155.
*       On entry DE = addr. of password buffer
*
50D1 21FFFF     LD      HL,0FFFFH    Mask value used for encode
50D4 0608       LD      B,08H        Max.no.chars. to encode
50D6 7B         LD      A,E          A = LSB of password buffer
50D7 C607       ADD     A,07H        Add 7 to position to end
50D9 5F         LD      E,A          of buffer. Must test for CARRY
50DA 3001       JR      NC,50DDH     And bump MSB if true
50DC 14         INC     D            Bump MSB of password buffer
50DD 1A         LD      A,(DE)       Fetch char. from password buffer
50DE D5         PUSH    DE           Save current buffer address
50DF 57         LD      D,A          Save char. to be encoded
50E0 5C         LD      E,H          E = Mark Value 1 (MV1)
50E1 7D         LD      A,L          A = Mark Value 2 (MV2)
50E2 E607       AND     07H          Reform MV2 by Exclusive ORing
50E4 0F         RRCA                 lower 3 bits and upper 3 bits
50E5 0F         RRCA                 Reposition lower 3 bits
50E6 0F         RRCA                 to upper 3 bits
50E7 AD         XOR     L            Exclusive OR lower and upper 3 bits
50E8 6F         LD      L,A          And save as new MV2
50E9 2600       LD      H,00H        Clear H to multiply new MV2 by 16
50EB 29         ADD     HL,HL        MV2 * 2
50EC 29         ADD     HL,HL        MV2 * 4
50ED 29         ADD     HL,HL        MV2 * 8
50EE 29         ADD     HL,HL        MV2 * 16
50EF AC         XOR     H            Combine upper 8 bits of MV2 * 16
50F0 AA         XOR     D            and partially reformed MV2 and
                                     current character
50F1 57         LD      D,A          Save upper 8 bits of encode
50F2 7D         LD      A,L          Preserve lower 8 bits of MV2 * 16
```

Listing Continued . . .

. . . Continued Listing

```
50F3 29          ADD     HL,HL        Compute MV2 * 32
50F4 AC          XOR     H            Combine upper 8 bits of MV2 * 32
50F5 AB          XOR     E            With lower 8 bits of MV2 * 16 and MV1
50F6 5F          LD      E,A          Save lower 8 bits of encode
50F7 EB          EX      DE,HL        HL = current encode
50F8 D1          POP     DE           Password buffer address
50F9 1B          DEC     DE           Skip down to next char.
50FA 10E1        DJNZ    50DDH        Loop till all characters encoded
50FC C9          RET                  Return to caller
```

Next, at **4E38** preparations are made for reading the HIT sectors. If a drive was specified, then only the specified drive will be read. If no drive was specified, then all drives will be searched starting with drive 0 and continuing through drive 4. The loop for reading the HIT sectors is controlled by the drive specification in **5154**.

The code from **4E38** to **4E5E** reads the HIT sectors and searches them for a matching hash code. Beginning at **4E38** the caller's drive specification is loaded into the C register. If no drive was specified, a -1 will be loaded. Otherwise, the drive number will be loaded.

At 4E3C/4E3F a simple procedure is executed to insure the C register has a positive value before the subroutine at **512E** is called to read the HIT sector. Before **512E** is called, another subroutine at **50FD** is called to determine if the drive to be read is available. If it is, the HIT sector is read and searched. If it is not, control goes to **4E54**, where the next unit number is computed.

The subroutine at **50FD** begins by sending a force interrupt command to the controller. This clears the controller so a reliable status can be read. Next, a subroutine in the nucleus at **4600** is called to select the drive to be read. Selecting the drive also forces motor on for the unit selected. If a diskette is mounted in the drive, it will begin spinning, and eventually the index status line will come true.

After selecting the drive, a count-down timer of approximately 8 seconds is loaded into the BC register. Next, a subroutine at **511E** is called to return the status of the index bit from the disk controller. The subroutine at **511E** decrements the BC counter before fetching the index status bit. If the counter goes to zero before index is detected, control returns to the caller of **50FD** with a non-zero (drive not available) status.

Beginning at **5109** and continuing through **5116**, three calls are made to **511E**. The first call at **5109** will loop if index is true, until it goes false. This is not a likely possibility, but it could happen if the index hole was under the sensor at the time the drive was selected. The second call at **510E** waits until index comes true. This means a drive is present and a diskette is mounted. The third call at **5113** waits for the index line to go false. This insures that the index true signal was not spurious.

If the timeout counter expires before all three conditions are met, the drive is assumed to be unavailable, and the appropriate status is returned to the caller of **50FD**.

Following the determination that a drive does exist, the code from **5119** to **511C** is executed before returning to the caller. This appears to be a test for drive ready, but it will always return a zero (good-status drive available). Upon entry, **512E** saves the caller's BC/DE register, calls a nucleus subroutine at **4B55** to load the directory track nuumber into the D register, loads the E register with a 1 (sector number), specifies a sector buffer address of **4D00**, and calls another nucleus subroutine at **4B35** to read the HIT sector. Upon return from **4B35**, the BC/DE registers are restored, and control returns to **4E48**, where an early exit is taken if a read error occurred. Following is the code used for these operations.

Figure 5.4 *SYS2 Code From 50FD to 511D*

```
*
*    Determine drive availability. Select drive and wait for
*    index. If no index detected within approx. 1 sec. there is
*    no drive, or the drive is not ready.
*
50FD 3ED0       LD      A,0D0H      Clear controller by sending it
50FF 32EC37     LD      (37ECH),A   a force interrupt command
5102 CD0046     CALL    4600H       Select unit in C-reg.
5105 C5         PUSH    BC          Save caller's BC
5106 015531     LD      BC,3155H    Count for approx.1 sec.
5109 CD1E51     CALL    511EH       Get disk status
510C 20FB       JR      NZ,5109H    Loop till index
510E CD1E51     CALL    511EH       Now
5111 28FB       JR      Z,510EH     wait 1 revolution
5113 CD1E51     CALL    511EH       And test for index again
5116 20FB       JR      NZ,5113H    Loop till 2nd index
5118 C1         POP     BC          Restore caller's BC
5119 07         RLCA                This code is meaningless
511A E680       AND     80H         and will always
511C 87         ADD     A,A         Return a zero in the A-register
511D C9         RET                 Return to caller
```

Figure 5.5 *SYS2 Code From 511E to 512D*

```
*
*       Decrement count in BC while waiting for INDEX. If count
*       goes to zero before INDEX detected, return to caller of
*       50FD with a non-zero status.
*
511E 0B          DEC     BC              Decrement count
511F 78          LD      A,B             Combine both halves
5120 B1          OR      C               Set status flag
5121 2806        JR      Z,5129H         .   Go if count reached zero (error)
5123 3AEC37      LD      A,(37ECH)       .   Else get click status
5126 CB4F        BIT     01H,A           .   Test for index
5128 C9          RET                     .   And return to caller
5129 C1          POP     BC              .   Clear stack
512A C1          POP     BC              Clear stack
512B F601        OR      01H             Return to
512D C9          RET                     50FD caller w/error status
```

Figure 5.6 *SYS2 Code From 512E to 5140*

```
*
*      --- Read HIT sector for drive sqecified in C register ---
*
512E C5          PUSH    BC              Save caller's
512F D5          PUSH    DE              BC and DE
5130 CD554B      CALL    4B55H           Get directory into D reg.
5133 1E01        LD      E,01H           E = sector number for HIT
5135 21004D      LD      HL,4D00H        Buffer address
5138 CD354B      CALL    4B35H           Read HIT sector from directory track
513C D1          POP     DE              Restore caller's
513C C1          POP     BC              BC and DE
513D C8          RET     Z               If no error return with good status
513E 3E16        LD      A,16H           Else signal HIT read error
5140 C9          RET                     Then return to caller
```

Assuming no read error occurred, a loop from **4E40** to **4E52** is executed to search the sector buffer at **4D00** for a matching hash code. If a match is found, control goes to **4E6A**, otherwise two tests are performed. The first tests for a caller-specified drive number. If this test is true, control goes to **4E60**, where a status indicating 'file not found' is loaded and control is returned to the caller. The second test, assuming no drive was specified, tests if all drives have been searched. This test is performed by incrementing the unit number in the C register and comparing it to 4. When the drive number equals 4, all drives have been searched, and control is returned to the caller with a status of file not found. The code used for these operations is shown below.

Figure 5.7 *SYS2 Code From 4E12 to 4E68*

```
*
*                       Begin OPEN Processing
*
4E12 CD9648     CALL    4896H           Save regs. setup return sequence
4E15 78         LD      A,B             A = Record length
4E16 32AF4F     LD      (4FAFH),A       Save record length
4E19 22C04F     LD      (4FC0H),HL      Callers sector buffer address
4E1C DDE5       PUSH    IX              DCB addr. to HL via stack
4E1E E1         POP     HL              HL = DCB addr.
4E1F CD2750     CALL    5027H           Move file name to local DCB (515D)
4E22 C0         RET     NZ              Exit if illegal chars. found
4E23 215D51     LD      HL,515DH        HL = Start of local DCB
4E26 CD9B50     CALL    509BH           Compute hash code for filename
4E29 324E4E     LD      (4E4EH),A       Save hash code
4E2C 115551     LD      DE,5155H        Addr. of password buffer
4E2F CDD150     CALL    50D1H           Get encode of password into HL
4E32 226851     LD      (5168H),HL      And save in
4E35 226A51     LD      (516AH),HL      Local DCB
4E38 3A5451     LD      A,(5154H)       Get drive specification flag
4E3B 4F         LD      C,A             C = Default unit number (0)
4E3C 3C         INC     A               = FF if none given, else = drive no.
4E3D 2001       JR      NZ,4E40H        Jump if drive specified
4E3F 4F         LD      C,A             Else begin with drive 0, and
4E40 CDFD50     CALL    50FDH             .  see if drive in C reg. present
4E43 200F       JR      NZ,4E54H          .  Jump if drive not present
4E45 CD2E51     CALL    512EH             .  Drive found, get table in 4D00
4E48 C0         RET     NZ                .  Exit if error during read
4E49 7E         LD      A,(HL)            .  Get hash code from HIT
                                          .  just read
4E4A B7         OR      A                 .  Set status flags for hash code
4E4B 2804       JR      Z,4E51H           .  Jump if slot empty
4E4D FE00       CP      00H               .  Compare hash from HIT
                                          .  to computed
4E4F 2819       JR      Z,4E6AH           .  Jump if hash codes match
4E51 2C         INC     L                 .  Else bump to next loc.
                                          .  in hash tab.
4E52 20F5       JR      NZ,4E49H          .  Jump if hash tab. not finished
4E54 3A5451     LD      A,(5154H)         .  No match. File not found,
                                          .  search next drive
4E57 3C         INC     A                 .  Unless drive was specified
4E58 2006       JR      NZ,4E60H          .  If drive specified
                                          .  file not found
4E5A 0C         INC     C                 .  Else bump drive number
4E5B 79         LD      A,C               .  Drive to A-reg. so we can test
4E5C FE04       CP      04H               .  if all drives were searched
4E5E 38E0       JR      C,4E40H           .  Jump if all drives
                                          .  not searched
4E60 3E18       LD      A,18H           Error code for file not found
4E62 B7         OR      A               Set status flags
4E63 C9         RET                     Return to caller
```

Listing Continued . . .

. . . Continued Listing

```
*
*               Clear Stack and Prepare to Search HIT
*
4E64 C1          POP     BC             Clear stack
4E65 E1          POP     HL             Clear stack
4E66 C1          POP     BC             Restore drive no. to C-reg.
4E67 E1          POP     HL             Restore current hash sector index
4E68 18E7        JR      4E51H          Go test next hash value
```

At **4E6A**, the directory sector containing the name will be read. When this code is entered, the HL register contains the address of the matching hash code in the HIT sector. The L register is an index into the HIT sector buffer and also contains the sector number minus two for the directory sector containing the file entry in bits 0 - 3.

A read for the directory sector is issued at **4E6D**. On return, the sector resides in the buffer at **4200**, and the HL register points to an advanced starting position in the buffer where the file name may be found. If no errors occurred during the read, control passes to **4E75**; otherwise, control is returned to the caller of SYS2 with an appropriate error code in the A register. The code used for these operations is given below.

Figure 5.8 *SYS2 Code From 4E6A to 4E74*

```
*
*               Read Directory Sector Containing File Entry
*                       Matching Hash Code Found
*
4E6A E5          PUSH    HL             Save hash code address
4E6B C5          PUSH    BC             Save B/unit number
4E6C 45          LD      B,L            B = Addr. of entry in directory
4E6D CDC14A      CALL    4AC1H          Read directory sector into 4200
4E70 2803        JR      Z,4E75H        Continue if no error
4E72 C1          POP     BC             Else clear
4E73 E1          POP     HL             Stack and
4E74 C9          RET                    Return to caller w/error status
```

At **4E75**, the directory entry pointed to by the HL register is compared to name of the file being opened. If they do not match, or if the directory entry is not a primary entry, control goes to **4E64** and from there to **4E51**.

At **4E64**, two pushes are cleared from the stack, the BC and HL registers are restored to the unit number and current index into the HIT sector buffer at **4D00**. Control then passes to **4E51**, where the next hash entry is compared to the hash value for the current file name.

This code permits two files to have the same hash value.

If the directory entry is a primary entry and the file name matches, control will fall through to **4E8C**. At that point, the unit number (**5154**) is updated to reflect the current drive, the HL register is restored to the starting address for the file in the directory sector, two pushes are cleared from the stack, and the unit number and directory entry address are saved on the stack.

Next, the permission flags are isolated and copied to the C register, the HL register is adjusted to point to the password, and the encode of the password is subtracted from 24,994. Finally, control passes to **4ECF**, and from there to **4FA7**. The following code is used to search the directory sector.

Figure 5.9 *SYS2 Code From 4E75 to 4EAC*

```
*
*          ---  Locate File Entry io Directory Sector  ---
*
4E75 E5          PUSH    HL              Save addr. of file entry
4E76 C5          PUSH    BC              And dir index, unit number
4E77 CB7E        BIT     Ø7H,(HL)        Test if primary directory entry
4E79 2ØE9        JR      NZ,4E64H        Jump if not primary entry
4E7B 3EØ5        LD      A,Ø5H           Offset to name in dir
4E7D 85          ADD     A,L             Add 5 to addr. of entry
4E7E 6F          LD      L,A             HL = Addr. of name in directory
4E7F 115D51      LD      D ,515DH        DE = Addr. of local DCB
                                         (contains name)
4E82 Ø6ØB        LD      B,ØBH           Max No. of chars. to match
4E84 1A          LD      A,(DE)          A = Char. from local DCB
4E85 BE          CP      (HL)            Compare to Char. in directory sector
4E86 2ØDC        JR      NZ,4E64H        Go to next entry if no match, else
4E88 23          INC     HL              Bump directory entry address and
4E89 13          INC     DE              Local DCB address
4E8A 1ØF8        DJNZ    4E84H           Loop till all chars. in name matched
*
*              Filename Located in Directory Sector
*         Isolate Access Permissions and Password Encodes
*
4E8C Cl          POP     BC              Restore unit number
4E8D 79          LD      A,C             Move unit number to A reg.
4E8E 325451      LD      (5154H),A       Save unit number
4E91 El          POP     HL              HL = directory entry sector addr.
4E92 Fl          POP     AF              Clear stack
4E93 Fl          POP     AF              Clear stack
4E94 C5          PUSH    BC              Save index/unit numcer
4E95 E5          PUSH    HL              Addr. of directory entry
                                         containing file
4E96 7E          LD      A,(HL)          Get first byte of directory entry
4E97 E6Ø7        AND     Ø7H             Isolate permission bits
4E99 4F          LD      C,A             And save in C
4E9A Ø6ØØ        LD      B,ØØH           Permission flags - no restriction
```

Listing Continued . . .

. . . Continued Listing

```
4E9C 3E10        LD      A,10H         Offset to password
4E9E 85          ADD     A,L           Gives addr. of update password
4E9F 6F          LD      L,A           HL = Addr. of password in dir entry
4EA0 ED5B6A51    LD      DE,(516AH)    Encode of password
4EA4 E5          PUSH    HL            Save addr. of dir (16) entry
4EA5 21A261      LD      HL,61A2H      HL = 24,994
4EA8 AF          XOR     A             Clear carriage
4EA9 ED52        SBC     HL,DE         24,994 - encode of password
4EAB E1          POP     HL            Restore addr. of password
4EAC 1821        JR      4ECFH         Go fill in DCB.
```

At **4FA7**, the DCB is initialized. After initializing the working registers, the DCB is flagged as open (at **4FAB**), the permission flags and I/0 type flags (physical or logical) are merged and saved in the second byte of the DCB (**4FAE-4FB9**). Next, byte 3 of the DCB is initialized to zero, the sector buffer address is stored in bytes 4 and 5, and the next record number and overflow pointer are saved at DCB plus 7 and 8.

Continuing at **4FCD**, the EOF byte offset is copied from the directory entry to the ninth byte of the DCB, the record length is copied from the directory entry, the next record nuumber is set to zero, and the EOF sector is copied from the directory to the DCB.

The code from **4FE5** to **4FE8** copies the number of sectors in the file (file size) from the directory to the DCB. The code from **4FEB** to **501B** copies the GAP's from the directory to the DCB and computes the sum of the granules addressable through each GAP. This is done for the benefit of the disk I/0 system, which uses the granule totals to determine which GAP contains a particular record. For more information about how the GAP's and the granule sums are used, see Section 3.4.5.

The track for the first GAP is copied from the directory sector buffer to the DCB at **4FED**. The granule count for the first GAP is copied at **4FF2/4FF3**. The granule count for the first GAP is also loaded into the BC register pair, which serves as granule accumulator for the loop at **4FFC** to **5019**.

This loop copies the remainder of the GAP's from the directory to the DCB. Before any GAP is moved, a test for end of GAP or overflow GAP pointer is performed at **4FFE/4FFF**. If either condition is true, control goes to **501C**, where the GAP chain is filled out.

If neither is true, the accumulated granule count (in the BC) is copied to the DCB (following the previous GAP), the starting track number and granule count for the next GAP is then copied, and the number of granules for the current GAP is added to the granule total in the BC registers. This loop is repeated until the end of the GAP's in the primary entry is reached.

When the last of the GAP's has been copied the DCB initialization is complete; the OPEN or INIT operation is also complete and control returns to the caller of SYS2. Following is the code used to initialize a DCB.

Figure 5.10 *SYS2 Code From 4FA7 to 5026*

```
*
*       Fill out DCB. Copy portions of primary entry to DCB. On
*        entry DE = addr of directory entry, HL = addr of DCB.
*
4FA7 EB          EX      DE,HL        DE addr. of directory entry
4FA8 DDE5        PUSH    IX           Prepare to load HL
4FAA E1          POP     HL           HL = addr. of caller's DCB
4FAB 3680        LD      (HL),80H     Signal file open
4FAD 23          INC     HL           Bump to 2nd byte of DCB
4FAE 3E00        LD      A,00H        A = record length
4FB0 B7          OR      A            Set status flags for record length
4FB1 3E00        LD      A,00H        Load access flags (put here by 4ED1)
4FB3 2802        JR      Z,4FB7H      Jump if rec. length is
                                      zero (phys. I/O)
4FB5 F680        OR      80H          Else, signal logical I/O
4FB7 F620        OR      20H          Force read on first access
4FB9 77          LD      (HL),A       Save access flags, type of I/O
4FBA 23          INC     HL           Bump to 3rd byte of DCB
4FBB 3600        LD      (HL),00H     and store a zero (Not used)
4FBD 23          INC     HL           Bump to 4th byte of DCB
4FBE D5          PUSH    DE           Save original DCB addr.
4FBF 110000      LD      DE,0000H     Sector buffer addr. to DE
4FC2 73          LD      (HL),E       LSB of buffer addr. to DCB
4FC3 23          INC     HL           Bump to 5th byte of DCB
4FC4 72          LD      (HL),D       MSB of buffer to DCB
4FC5 23          INC     HL           Bump to 6th byte of DCB
4FC6 D1          POP     DE           Restore dir. entry addr.
4FC7 3600        LD      (HL),00H     Offset to end of cusrent record
4FC9 23          INC     HL           Bump to 7th byte of DCB
4FCA 71          LD      (HL),C       Save drive no.
4FCB 23          INC     HL           Bump to 8th byte of DCB
4FCC 70          LD      (HL),B       Save overflow pointer
4FCD 23          INC     HL           Bump to 9th byte of DCB
4FCE 3E03        LD      A,03H        Index to EOF byte in directory
4FD0 83          ADD     A,E          Add to LSB of addr. of
4FD1 5F          LD      E,A          base of directory sector
4FD2 1A          LD      A,(DE)       Load EOF byte offset from dir entry
4FD3 77          LD      (HL),A       Save EOF byte offset
4FD4 23          INC     HL           Bump to 10th byte of DCB
4FD5 13          INC     DE           Bump to next by of directory
4FD6 3AAF4F      LD      A,(4FAFH)    Load record length
4FD9 77          LD      (HL),A       Move record length to DCB
4FDA 23          INC     HL           Bump to 11th byte of DCB
4FDB 3600        LD      (HL),00H     Clear LSB of next record no.
4FDD 23          INC     HL           Bump to MSB
4FDE 3600        LD      (HL),00H     Clear MSB of next record no.
4FE0 23          INC     HL           Bump to 13th byte of DCB
4FE1 3E10        LD      A,10H        Index to EOF sector number
4FE3 83          ADD     A,E          Add to base of dir sector addr.
4FE4 5F          LD      E,A          Form addr. of 16th byte in DE
```

Listing Continued . . .

. . . Continued Listing

```
4FE5 1A          LD      A,(DE)       Fetch EOF sector (LSB)
4FE6 77          LD      (HL),A       And save in DCB
4FE7 23          INC     HL           Bump to 14th byte in DCB
4FE8 CD8051      CALL    5180H        Copy MSB of number of sectors
4FEB 23          INC     HL           Bump to 1st track addr. in DCB
4FEC 13          INC     DE           Bump to 1st trk. addr. in directory
4FED 1A          LD      A,(DE)       Fetch starting track number
4FEE 77          LD      (HL),A       Save in DCB
4FEF 23          INC     HL           Bump to gran count for 1st GAP in
4FF0 13          INC     DE           Dir sector and LSB of accumulated
                                      granule in DCB
4FF1 1A          LD      A,(DE)       Fetch # of granules from dir entry
4FF2 77          LD      (HL),A       Copy it to DCB
4FF3 23          INC     HL           Bump to track # for 2nd GAP
4FF4 E61F        AND     1FH          Isolate no.of consecutive
                                      granules -1 for 1st GAP
4FF6 3C          INC     A            Gives no. of consecutive grans
4FF7 4F          LD      C,A          BC will contain total grans
4FF8 0600        LD      B,00H
4FFA 3E04        LD      A,04H        No. of extant pairs remaining
4FFC F5          PUSH    AF           .   Save count.
4FFD 13          INC     DE           .   Bump to next GAP
4FFE 1A          LD      A,(DE)       .   Fetch next GAP from directory
4FFF FEFE        CP      0FEH         .   Look for end of GAP or
                                      .   overflow pointer
5001 3019        JR      NC,501CH     .   Jump if end or overflow pointer
5003 71          LD      (HL),C       .   Save LSB of total grans for
5004 23          INC     HL           .   previous GAP
5005 70          LD      (HL),B       .   Save MSB of total
5006223          INC     HL           .   Bump to first pos. in next GAP
5007 1A          LD      A,(DE)       .   Fetch track for next GAP
5008 77          LD      (HL),A       .   And save in DCB
5009 23          INC     HL           .   Bump to 2nd byte of GAP in DCB
500A 13          INC     DE           .   Bump to count in dir. sector
500B 1A          LD      A,(DE)       .   Get count of assigned grans
500C 77          LD      (HL),A       .   Save in DCB
500D 23          INC     HL           .   Bump to LSB of totals in DCB
500E E61F        AND     1FH          .   Isolate count from starting
                                      .   sector number
5010 3C          INC     A            .   Get true granule count
5011 81          ADD     A,C          .   Form new total granules
5012 4F          LD      C,A          .   Accessible thru
5013 78          LD      A,B          .   GAPS thus far
5014 CE00        ADC     A,00H        .   capture carry in of overflow
5016 47          LD      B,A          .   BC = Total granules accessible
5017 F1          POP     AF           .   Restore no. of GAP's to copy
5018 3D          DEC     A            .   Have we copied all GAP's
5019 20E1        JR      NZ,4FFCH     .   Jump if not
501B C9          RET                  Else ret to OPEN/INIT caller
*
*       End of GAP's in directory entry have been reached.
*       Fill remainder of DCB GAP area with bytes of FF.
*
501C F1          POP     AF           Number of unfilled entries
501D 07          RLCA                 Times four
```

Listing Continued . . .

. . . Continued Listing

```
501E 07          RLCA                    gives byte count
501F 47          LD       B,A            Move to B reg.for loop control
5020 36FF        LD       (HL),0FFH      Signal end of GAP's
5022 23          INC      HL             Bump to next entry in DCB
5023 10FB        DJNZ     5020H          Loop till DCB filled
5025 AF          XOR      A              Signal good status
5026 C9          RET                     Return to caller
```

5.2 Request Code 20 (INIT)

A request code of 20 selects INIT processing. This processing is very similar to OPEN processing, and shares some of its code. It consists of the following steps:

1. Use OPEN code to compute hash and attempt to locate the file.
2. If the file was located at Step 1, return to caller.
3. Otherwise, find an available slot in a HIT sector, assign slot to the file and rewrite the updated HIT sector.
4. Read the directory sector associated with the ordinal assigned in the HIT sector.
5. Initialize the file name entry in the directory sector and rewrite the updated sector.
6. Construct the DCB.

The code for INIT processing begins at **4ED8**. As with OPEN, the first call is to a nucleus subroutine at **4896** to preserve the register context and set up an exit sequence. Next, OPEN is called. If the file already exists, a zero status will be returned, and control will be returned to the caller because all processing has been completed.

When a non-zero status is returned, its exact value must be tested. A status of **18** means the file was not found and a directory entry for it must be created. Any other non-zero status indicates a non-recoverable error during OPEN processing. In that case, control returns to the caller with an error status.

Starting at **4EE2** preparations are made to assign a HIT slot and initialize a directory entry. The code at **4EE2** modifies the instruction at **4F1C** to store an 'assigned' byte in the first byte of a directory entry. Following the code at **4EE2** is a test for a user-specified drive number. Usage of **5154** and a description subroutine of the subroutine at **50FD** can be found in Section 5.1.

The code from **4EE7** to **4EF4** will select the user-specified drive, or find an available one. At **4EF6**, a subroutine at **512E** is called to read the HIT sector for the unit selected.

The subroutine at **512E** begins by saving the BC/DE register set. Next, a nucleus subroutine at **4B55** is called to return the directory track number for the unit in the C register. After that, the E register is loaded with the sector number for the HIT sector, a buffer address of **4D00** is loaded into the HL register, and a nucleus subroutine at **4B35** is called to read the HIT sector. On return from **4B35**, the BC/DE registers are restored, and control is returned to the caller with the read status in the A register.

Back at **4EF9**, the read status is tested. Assuming a good status, the subroutine at **50AB** is called to locate an available slot in the HIT sector.

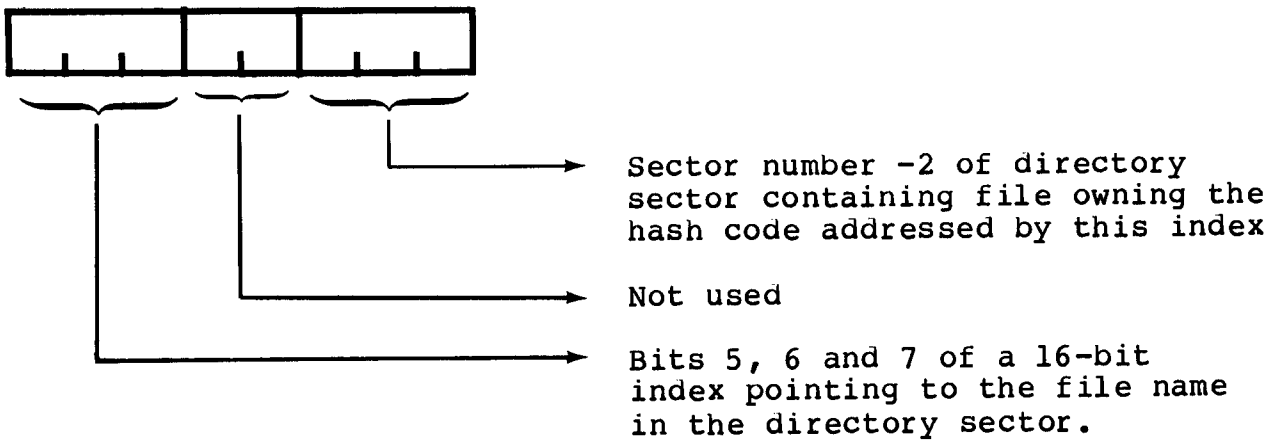
The subroutine at **50AB** will search the HIT sector for an available slot within the boundaries of **40 - 47, 60 - 67,E0 - E7**. An arbitrary starting point within the sector buffer is chosen by masking the contents of **4040** (updated every 25 milliseconds) and forcing it into the range **40 - 47**, etc. Next, at **50B7** a call to a subroutine at **50BD** is made to locate an available slot. If the call is unsuccessful, **50BD** in re-executed by falling into it with a starting point of **40** in the HIT buffer.

50BD searches the HIT buffer looking for available (zero) slot. On entry, the HL register contains the starting position with the HIT buffer where the search will begin. The search consists of two loops. The first loop goes from **50BD** to **50C4**. Each time an unavailable entry is found, this loop bumps the index by **20**. This loop terminates when an available slot has been found, or the index exceeds **E7**.

The second loop goes from **50BD** to **50CC**. When the index overflows from the first loop, the second loop is entered. When entered, the A register will contain a value (at **50C6**), and the result is tested for overflow out of the lower three bits (least significant digit greater than 7). If no overflow occurs, control returns to the first loop. This varies the last digit of the index by 1. Eventually, if no slot is found, the last digit will exceed 7, and control will be returned to the caller of **50AB** with a non-zero (no slot available) status.

HIT slots are assigned for the ordinals **40 - 47, 60 - 67,E0 - E7**. The indices for **00 - 07**, and **20 - 27** are reserved for system use. The HIT index can be used to derive the sector number, and the file position within the directory sector. It has the following format:

Figure 5.11 *HIT Index Description*



Because of the restriction in assigning slot indexes, slots for the system files BOOT/SYS, DIR/SYS, SYS0/SYS, ... SYS6/SYS must be assigned manually. This restriction is not arbitrary. Recall that the overlay load request specifies a directory sector number and an index of 0 or 1 into the directory sector. The HIT index restriction guarantees that the first and second entry in the directory sector will not be assigned to user files, and will always be available for system files. This also restricts the number of slots available for user files to 48.

Note, also, that the entire HIT sector is searched during OPEN and INIT processing. If it were not, there would be no way to access the system files using TRSDOS utilities. Shown below is the code used for locating an available slot in the HIT sector.

Figure 5.12 *SYS2 Code From 50AB to 50D0*

```
*
*              Find an available hole in HIT sector.
*
50AB 3A4040      LD      A,(4040H)     Choose a random number
50AE E6E7        AND     0E7H          Force it to 00-07, 20-27, 40-47,
50B0 FE40        CP      40H           Then test if below 40
50B2 3002        JR      NC,50B6H      Jump if not, else
50B4 F680        OR      80H           force it 80-87, A0-A7, C0-C7, or E0-E7
50B6 6F          LD      L,A           Form beginning search address in HL
50B7 CDBD50      CALL    50B           Search HIT, look for available slot
50BA C8          RET     Z             Exit if one found
50BB 2E40        LD      L,40H         Search again from start of table
```

Listing Continued . . .

. . . Continued Listing

```
50BD 7E          LD      A,(HL)          .   Get a HIT entry
50BE B7          OR      A               .   Set status floor
50BF C8          RET     Z               .   Exit if avail.byte found
50C0 7D          LD      A,L             .   Bump to first
50C1 C620        ADD     A,20H           .   Byte of next set of entries
50C3 6F          LD      L,A             .   Reform address
50C4 30F7        JR      NC,50BDH        .   Jump if end of table not reached
50C6 C641        ADD     A,41H           Restart search at 2, 3, ... of a set.
50C8 6F          LD      L,A             Reform addr in range 4X-47, 6X-67,...
50C9 E6E7        AND     0E7H            where X=1,2,...7
50CB BD          CP      L               If addr LSB = E7, found end of table
50CC 28EF        JR      Z,50BDH         Jump if not end of table
50CE F601        OR      01H             Signal no available entries
50D0 C9          RET                     Return to caller
```

After the HIT buffer has been searched, control returns to **4EFD**, where the results of the search are tested. If a slot was found, control passes to **4FOE**. If no slot was found, the HIT sector on the next available drive is read and searched (unless a drive was specified in the INIT call). If no slot is available in any HIT sector, control returns to the INIT initiator with a directory full status.

Control goes to **4FOE** when a slot has been found. The HL register will contain the address of the available slot; **4E4E** contains the hash code (computed and stored during OPEN processing). After storing the hash code in the HIT slot, the HIT sector is rewritten to the disk via a call to **5141**.

5141 saves the caller's BC/DE register, calls the nucleus subroutine **4B55** to fetch the directory track number for the unit in register C, and calls **45CF** to rewrite the HIT sector. Control returns to **4F16** with the rewrite status in the A register. Below is the code used for the subroutine at **5141**.

Figure 5.13 *SYS2 Code From 5141 to 5153*

```
*
*     --- Write HIT sector for drive specified in C register ---
*
5141 C5          PUSH    BC              Save caller's
5142 D5          PUSH    DE              BC and DE registers
5143 CD554B      CALL    4B55H           Get directory track number
                                         for unit in C-reg.
5146 1E01        LD      E,01H           Specify sector 1 (HIT sector)
5148 21004D      LD      HL,4D00H        Address of HIT sector buffer
514B CDEF46      CALL    46EFH           Write HIT sector
514E D1          POP     DE              Restore caller's
514F C1          POP     BC              BC and DE
5150 C8          RET     Z               Exit if no error during write
5151 3E17        LD      A,17H           Else signal HIT write error
5153 C9          RET                     And return to caller
```

If no error occurred during the write operation, the nucleus subroutine at **4AC1** is called to read the directory sector associated with the slot in the HIT. The directory sector number and entry within the sector are derived from the index for the slot assigned. That index was preserved in the B register at **4F0E** after the slot assignment was made by the call to **50AB** at **4EFA**.

Upon return from the directory read call at **4F17**, the HL register contains the address of the directory entry to be assigned. The instruction at **4F1C** modifies the first byte of the directory to indicate the entry is assigned. The code from **4F1F** to **4F25** clears the permission flags and the E0F offset byte. The record length is saved at **4F28**, and the name is copied from the internal buffer (**515D**) by the LDIR instruction at **4F35**. The number of records in the file are zeroed at **4F38** - **4F3D**, and the GAP's are initialized to **FF**'s by the loop for **4F3E** to **4F43**.

After the directory has been initialized, it is rewritten by a call to the nucleus subroutine at **4AD6**. As with the read call, the B register contains the HIT slot index, which is used to derive the directory sector to be written.

Following the directory sector rewrite, the subroutine at **4AF7** is called to fill out the caller's DCB. This subroutine is described in detail in Section 5.1. Shown below is the controlling code used during INIT processing:

Figure 5.14 *SYS2 Code From 4ED8 to 4F4F*

```
*
*                          INIT Processing
*
4ED8 CD9648      CALL    4896H          Save regs, setup exit sequence
4EDB CD154E      CALL    4E15H          Make OPEN call
4EDE C8          RET     Z              Return if file found
4EDF FE18        CP      18H            Test for file not found. Fall
                                        through if file not there
4EE1 C0          RET     NZ             Return if some other error
*
*             ---  File Does not Exist. Create Entry  ---
*
4EE2 3E10        LD      A,10H          Byte 0 of dir to be assigned
4EE4 321D4F      LD      (4F1DH),A      Entry assigned. Unrestricted access
4EE7 3A5451      LD      A,(5154H)      Get drive specification
4EEA 4F          LD      C,A            Save it in C-reg.
4EEB 3C          INC     A              Test if drive no. was specified
4EEC 2001        JR      NZ,4EEFH       Jump if unit specified
4EEE 4F          LD      C,A            Current unit number to C-reg.
```

Listing Continued . . .

. . . Continued Listing

```
4EEF CDFD50      CALL    50FDH          .   Go retest unit
4EF2 200B        JR      NZ,4EFFH       .   Jump if drive not present
4EF4 3809        JR      C,4EFFH        .   Jump if drive not ready
4EF6 CD2E51      CALL    512EH          .   Read hash table
4EF9 C0          RET     NZ             .   Exit if error during HIT read
4EFA CDAB50      CALL    50ABH          .   Find available slot in HIT
4EFD 280F        JR      Z,4F0EH        .   Go if slot for hash code found
4EFF 3A5451      LD      A,(5154H)      .   None left. Get drive spec.
4F02 3C          INC     A              .   Was a drive specified
4F03 2006        JR      NZ,4F0BH       .   Jump if yes (directory is full)
4F05 0C          INC     C              .   Bump to next drive
4F06 79          LD      A,C            .   So we can test drive number
4F07 FE04        CP      04H            .   Have all drives been searched
4F09 38E4        JR      C,4EEFH        .   If not, loop - else return
4F0B 3E1A        LD      A,1AH          space full error status
4F0D C9          RET                    Return to caller
4F0E 45          LD      B,L            B = Directory sector no
4F0F 3A4E4E      LD      A,(4E4EH)      Get hash code
4F12 77          LD      (HL),A         Save in HIT
4F13 CD4151      CALL    5141H          Write updated HIT
4F16 C0          RET     NZ             Ret. if error during write
4F17 CDC14A      CALL    4AC1H          Read directory sector that will
                                        contain file entry
4F1A C0          RET     NZ             Exit if error during read
4F1B E5          PUSH    HL             Save starting addr. of dir entry
4F1C 3600        LD      (HL),00H       Clear access control
4F1E 23          INC     HL             Bump to byte 2
4F1F 3600        LD      (HL),00H       Clear overflow pointer
4F21 23          INC     HL             Bump to byte 3
4F22 3600        LD      (HL),00H       Clear reserved byte
4F24 23          INC     HL             Bump to Byte 4
4F25 3600        LD      (HL),00H       Clear EOF offset byte
4F27 23          INC     HL             Bump to 5th byte
4F28 3AAF4F      LD      A,(4FAFH)      Get record length
4F2B 77          LD      (HL),A         And save in directory entry
4F2C 23          INC     HL             Bump to 6th byte in directory
4F2D C5          PUSH    BC             Preserve BC.
4F2E 010F00      LD      BC,000FH
4F31 EB          EX      DE,HL          DE = dir addr. for name
4F32 215D51      LD      HL,515DH       Internal DCB addr.
4F35 EDB0        LDIR                   Copy file name to dir entry
4F37 EB          EX      DE,HL          HL = Byte 15 of directory sector
4F38 3600        LD      (HL),00H       Clear EOF sector number (LSB)
4F3A 23          INC     HL             Bump to 16th byte of dir sector
4F3B 3600        LD      (HL),00H       Clear EOF sector number (MSB)
4F3D 23          INC     HL             Bump to 17th byte of directory
4F3E 060A        LD      B,0AH          Number of bytes to initialize
4F40 36FF        LD      (HL),0FFH      .   Store an FF in GAP field of
4F42 23          INC     HL             .    dir entry. Bump to next entry
4F43 10FB        DJNZ    4F40H          .   loop till all 10 GAPs initialized
4F45 C1          POP     BC             Restore HIT slot index used to
                                        derive dir. sector #.
4F46 CDD64A      CALL    4AD6H          Write updated directory sector
4F49 E1          POP     HL             Restore directory entry addr. to HL
4F4A C0          RET     NZ             Return if error
```

Listing Continued . . .

. . . Continued Listing

```
4F4B CDA74F      CALL    4FA7H       Go initialize DCB in user's area
4F4E 37          SCF                 Signal file not previously opened
4F4F C9          RET                 Return to caller
```

5.3 Request Code 30 (CREATE OVERFLOW ENTRY).

Overflow directory entries are created when the number of GAP's assigned to a file exceeds the GAP area in the directory entry for the file. There is no limit to the number of overflow entries that can be assigned. The GAP area for primary and overflow directory entries is identical.

Requests to create an overflow entry are made automatically when the system detects that the GAP area in the directory is full. This situation can only occur during a write operation. There is no way a user can make an explicit call to create overflow entries.

The code for assigning overflow entries begins at **4F50**. It is similar to both the OPEN and INIT processing previously described, and uses many of the subroutines called by those functions. There is one restriction which applies to overflow assignment that was not necessarily true with OPEN and INIT. The overflow sector must be assigned on the diskette where the file resides. The nearest parallel with OPEN and INIT processing would be the case where the call specified a particular drive.

The procedure used for assigning an overflow directory entry is as follows:

1. Read HIT sector from drive where file resides and find an available index.
2. Assign the index (use hash code for primary index), and rewrite the updated HIT. This is done for bookkeeping purposes to indicate a directory entry has been used.
3. Read the directory sector associated with index assigned at Step 2.
4. Initialize the entry in the directory sector as an overflow entry. Store pointer to previous entry and zero GAP area.
5. Write updated directory sector.
6. Read previous entry (usually the primary, but not necessarily), and modify entry to point to overflow entry.
7. Rewrite the updated previous directory entry.

Beginning at **4F50**, the drive where the file resides is loaded into the C register. This register will be preserved throughout the overflow processing. At **4F53** a subroutine at **512E** is called to read the HIT sector. This subroutine is discussed in detail in Section 5.1.

Following the read of the HIT sector, the hash index for the primary entry is loaded into the A register, and the subroutine at **50B6** is called to find the next available index. **50B6** is an advanced entry point into the subroutine at **50AB**, which is discussed in Section 5.1.

On return, the L register contains the next available index into the HIT. This index is copied to the B register, where it will be used for sector number computations by nucleus subroutines called to read and write the directory sector. It is also moved to **4AF2**, where it will be copied into the last byte of the primary directory entry (this index doubles as a pointer to the overflow directory entry).

Next, at **4F65**, the index for the primary entry is formed in the DE register, so that the hash code can be loaded at **4F69**. The hash code loaded is then stored at the new index by the instruction at **4F6A**. Notice that the same hash code is stored in the HIT twice. One index points to a primary directory; the other one points to an overflow directory.

Continuing on, at **4F6B** the updated HIT sector is rewritten (a hash index was assigned), and at **4F6F** the nucleus subroutine is called to read the directory sector for the overflow entry. Upon return, the HL register contains the address of the entry in the directory sector buffer.

The entry is initialized as an overflow entry at **4F73**; the hash index for the primary entry is moved to the second byte of the overflow entry. The name and password areas are cleared to zeros, and the GAP area is initialized to **FF**'s. After initialization, a nucleus subroutine at **4AD6** is called to rewrite the initialized overflow entry.

Next, at **4F92**, the hash index for the primary entry is retrieved from a byte in the nucleus, and the primary entry is read. The entry is then updated to contain a **FE** in the next to last byte, and the hash index for the overflow entry in the last byte. The modified primary entry is then rewritten and control returns to the SYS2 caller.

Practices such as these insure that the system will always be a single user system. Consider what would happen, for example, if TRSDOS was a multi-user system and one user was de-scheduled in SYS2 before he could retrieve the sector number from SYS0. Now, suppose another user is scheduled, and a call is made which causes the sector number to be modified. When the first user is re-scheduled, the sector number he was expecting has been changed. There are a number of ways to avoid this problem, the common one being to inhibit scheduling or interrupts, but a better solution is to avoid these practices. Below is the code used to create overflow entries.

Figure 5.15 *SYS2 Code From 4F50 to 4FA6*

```
*
*                Construct an Overflow Directory Entry
*
*        1. Find an available slot in HIT sector
*        2. Move file hash code to HIT slot located above
*        3. Write updated HIT
*        4. Read directory sector for HIT slot assigned
*        5. Initialize directory sector entry for HIT assigned
*           in step 1, as an overflow entry
*        6. Write directory sector containing overflow entry
*        7. Read directory sector containing primary entry
*        8. Update primary entry with overflow flag and pointer
*        9. Write updated primary entry
*
```

Listing Continued . . .

. . . Continued Listing

```
4F50 DD4E06      LD      C,(IX+06H)    C = drive number
4F53 CD2E51      CALL    512EH         Read HIT
4F56 C0          RET     NZ            Return if error during read
4F57 DD7E07      LD      A,(IX+07H)    A = Dir sector offset for this file
4F5A CDB650      CALL    50B6H         Find an available hole in HIT
4F5D 3E1E        LD      A,1EH         Error code for directory full
4F5F C0          RET     NZ            Return w/error if hash table full
4F60 45          LD      B,L           B = Next avail. loc. in HIT
4F61 7D          LD      A,L           Move it to A-reg.
4F62 32A24F      LD      (4FA2H),A     Save address of overflow entry
4F65 54          LD      D,H           D = MSB addr. of HIT buffer
4F66 DD5E07      LD      E,(IX+07H)    E = LSB addr. of HIT buffer
                                       (index for this file)
4F69 1A          LD      A,(DE)        Get hash code for file from HIT
4F6A 77          LD      (HL),A        Copy it to overflow entry
4F6B CD4151      CALL    5141H         Write updated HIT
4F6E C0          RET     NZ            Return if error during WRITE
4F6F CDC14A      CALL    4AC1H         Read dir sector for HIT slot assigned
4F72 C0          RET     NZ            Ret if directory read error
4F73 3690        LD      (HL),90H      Flag dir entry as assigned/overflow
4F75 2C          INC     L             Bump to 2nd byte of dir. entry
4F76 C5          PUSH    BC            Preserve unit number (C-reg.)
4F77 3ABE4A      LD      A,(4ABEH)     A = dir sector index for this file
4F7A 77          LD      (HL),A        Save as 2nd byte of overflow entry
4F7B 2C          INC     L             HL = 3rd byte of overflow entry
4F7C 0614        LD      B,14H         B = no. of bytes to zero
4F7E 3600        LD      (HL),00H      Zero a byte
4F80 2C          INC     L             Bump to next byte of overflow entry
4F81 10FB        DJNZ    4F7EH         Loop till 20 bytes cleared
4F83 E5          PUSH    HL            Save addr. of 1st GAP in entry
4F84 060A        LD      B,0AH         Number of bytes to initialize
4F86 36FF        LD      (HL),0FFH     Initialize
4F88 2C          INC     L             Bump to next byte
4F89 10FB        DJNZ    4F86H         Loop till all GAP's initialized
4F8B D1          POP     DE            DE = addr. of 1st GAP in entry
4F8C 13          INC     DE            Bump to number of grans (17th byte)
4F8D C1          POP     BC            Restore unit number to C-reg.
4F8E CDD64A      CALL    4AD6H         Write updated directory sector
                                       (contains overflow entry)
4F91 C0          RET     NZ            Exit if error during write
4F92 3ABE4A      LD      A,(4ABEH)     Get sector number for primary entry
4F95 47          LD      B,A           Move to B for nucleus read routine
4F96 CDC14A      CALL    4AC1H         Read primary entry sector
4F99 C0          RET     NZ            Exit if error during read
4F9A 7D          LD      A,L           HL = Addr of primary entry in sector
4F9B C61E        ADD     A,1EH         Position to 1st byte of 5th GAP
4F9D 6F          LD      L,A           HL = Directory entry (IE)
4F9E 36FE        LD      (HL),0FEH     Signal an overflow entry exists
4FA0 2C          INC     L             Bump to 2nd byte of 5th GAP
4FA1 3600        LD      (HL),00H      Save pointer to overflow entry
4FA3 CDD64A      CALL    4AD6H         Write updated primary entry
4FA6 C9          RET                   Return to caller
```

5.4 Description of Subroutine at 5027

This subroutine is called during OPEN processing to examine the file name and separate its components. The name can consist of: an 8-character file name, a 3 character extension, an 8-character password, and a 1-character drive specification. All components, except for the file name are optional.

Internal buffers in SYS2 are used to hold the file name components. An 8-byte buffer at **515D** holds the file name; an 8-byte buffer at **5155** holds the password; a 3-byte buffer at **5165** holds the extension; and the drive specification is kept at **5154**.

The code from **506F** to **5076** converts an ASCII digit to binary. This code is called to process the drive number if one has been specified.

Beginning at **5041**, a series of calls to **5081** is made to examine the file name and copy the components to their internal buffers. The subroutine at **5081** is called with a destination buffer address in the DE register, a source buffer address in the HL register, and the maximum number of characters to copy in the B register. Before a character is moved from the source to the destination address, it is verified as a letter A - Z, or a digit 0 - 9. Any other character is considered a terminator and causes control to be returned to the caller. If no illegal characters are detected, control is returned to the caller when the number of characters specified in the B register have been copied. When control is returned to the caller, the A register contains the last character loaded from source address.

Following the call to **5081** to copy the file name, the B register is compared to 8. This is a simple test to determine if any characters were copied. If none were, an error has occurred, and control is returned to the caller.

If a file name was copied, the terminating character in the A register is compared to the various delimiters (/, :, .), and depending on which one is found, control goes to **5055** (get extension), **5063** (get password), or **506F** (get drive specifications). After the processing for any of the file name options, the terminating character is compared to the delimiter for the remaining options. Notice that because of the way this code is structured, options must occur in a specific order; they cannot be intermixed or occur in random order.

The code from **506F** to **5076** converts an ASCII digit to binary. This code is called to process the drive number if one has been specified.

Control returns to the caller when an illegal character is detected, or all components of the file name have been processed. The code used for this subroutine follows:

Figure 5.16 *SYS2 Code From 5027 to 5080*

```
*
*     -- Copy caller's file name to internal DCB. Validate name --
*
5027 060B       LD      B,0BH          Number of chars. to copy
5029 115D51     LD      DE,515DH       Internal DCB address
502C 3E20       LD      A,20H          ASCII blank
```

Listing Continued . . .

. . . Continued Listing

```
502E 12          LD      (DE),A       Move blank to internal DCB
502F 13          INC     DE           Bump to next internal DCB addr.
5030 10FC        DJNZ    502EH        Loop till internal DCB blank filled
5032 0608        LD      B,08H        Max. number of chars in file name
5034 115551      LD      DE,5155H     Address of password buffer
5037 12          LD      (DE),A       Blank fill password buffer
5038 13          INC     DE           One byte at a time
5039 10FC        DJNZ    5037H        Loop till buffer blank filled
503B 3EFF        LD      A,0FFH       Signal no drive spec (default)
503D 325451      LD      (5154H),A    (Reversed if drive specified)
5040 115D51      LD      DE,515DH     Addr. of Internal DCB
5043 0608        LD      B,08H        No. of chars. to move
5045 CD8150      CALL    5081H        Move file name to buffer at 515D
5048 4F          LD      C,A          C = last char.in name
5049 78          LD      A,B          A = 8 - number of chars. copied
504A FE08        CP      08H          If = 8, then no. chars. were copied
504C 2004        JR      NZ,5052H     Jump if file name was legitimate
504E 3E13        LD      A,13H        Error code for illegal char.
5050 B7          OR      A            Set status flags
5051 C9          RET                  And return to caller
5052 79          LD      A,C          Get terminating character
5053 FE2F        CP      2FH          Test for /
5055 2008        JR      NZ,505FH     Jump if no extension
5057 116551      LD      DE,5165H     Addr of ext. area in internal DCB
505A 0603        LD      B,03H        Max. number of chars. in extension
505C CD8150      CALL    5081H        Copy extension
505F FE2E        CP      2EH          Was terminating char. a .
                                      (password follows)
5061 2008        JR      NZ,506BH     Jump if not
5063 115551      LD      DE,5155H     Addr of internal password buffer
5066 0608        LD      B,08H        Maximum no. of chars. in password
5068 CD8150      CALL    5081H        Copy password to internal PW buffer
506B FE3A        CP      3AH          Was terminating char. a : (drive
                                      specification follows)
506D 2010        JR      NZ,507FH     Jump if no
506F 7E          LD      A,(HL)       Get drive number
5070 D630        SUB     30H          ASCII to binary
5072 3804        JR      C,5078H      Jump if drive number not a digit
5074 FE04        CP      04H          Test for drive no. greater than 3
5076 3804        JR      C,507CH      Jump if drive no. in range(0-3)
5078 3E20        LD      A,20H        Error code for illegal drive no.
507A B7          OR      A            Set status flags for error.
507B C9          RET                  Return to caller
507C 325451      LD      (5154H),A    Save drive specification
507F AF          XOR     A            Signal good status (no error)
5080 C9          RET                  Return to caller
```

5.5 Hash Code Computation

The hash code subroutine begins at **509B** and ends at **50AA**. It is a straightforward hash that produces a reasonably random value between **01** and **FF**. It hashes 11 characters, regardless of the length of the file name.

There are no error exits from this subroutine. Control is returned to the caller with the hash code in the A register. Following is the code used for computing hash values:

Figure 5.17 *SYS2 Code From 509B to 50AA*

```
*
*     Compute hash code. Exclusive OR each character of file name
*     with the following character. Multiply result by 2 and use
*     as current character.
*
509B 060B       LD      B,0BH       Iteration count
509D 0E00       LD      C,00H       Initialize a mark value
509F 7E         LD      A,(HL)      .   Get a char from file name
50A0 23         INC     HL          .   Bump to next char. in name
50A1 A9         XOR     C           .   Exclusive OR with floating mask
50A2 07         RLCA                .   Result times 2
50A3 4F         LD      C,A         .   becomes new floating mask
50A4 10F9       DJNZ    509FH       .   Loop till 11 characters of
                                        file name masked
50A6 79         LD      A,C         A = Final mask value
50A7 B7         OR      A           Set status flags
50A8 C0         RET     NZ          OK if non-zero
50A9 3C         INC     A           Else force a value of 1
50AA C9         RET                 and exit
```

6

6.0 SYS3/SYS

SYS3/SYS is a continuation of the TRSDOS file management system. It is called from the nucleus in response to CLOSE and KILL requests.

SYS3 begins at **4E00** and ends at **5085**. The locations **5086** through **50FF** contain **FF**'s. The reason for this initialization is not clear. No part of SYS3 uses this region. The entry point for SYS3 is **4E00**.

Before branching to the CLOSE or KILL code, SYS3 performs several tests on both the DCB and the sector buffer addresses to insure they are within acceptable limits. Additional tests are made to verify the drive number in the DCB is valid and that the first track number is less than or equal to 35. If any of these tests fail, the operation is aborted, and an error code of 26 (file not opened) is returned.

The entry code at **4E00** begins by saving the DCB address, buffer address, and request code (IX, HL, and AF registers, respectively). Immediately after saving these registers, three tests are performed on the DCB. The first test, which begins at **4E04**, verifies the DCB is RAM resident by comparing the most significant byte of its address to **40**(RAM begins at **4000**). The next two tests insure the DCB address is below the end of memory (don't laugh, it could happen), and that the file has been opened. The following code is used for these operations:

Figure 6.1 *SYS3 Code From 4E00 to 4E14*

```
4E00 DDE5        PUSH    IX          Save DCB address
4E02 E5          PUSH    HL          Save buffer address
4E03 F5          PUSH    AF          Save request code
*
* Validate address of DCB and buffer. Validate drive
* number, HIT index and beginning track number
*
4E04 7A          LD      A,D         A = MSB of DCB addr.
4E05 FE40        CP      40H         Is DCB in system area?
4E07 385C        JR      C,4E65H     Error if yes
```

Listing Continued . . .

. . . Continued Listing

```
4E09 214A40      LD      HL,404AH      Addr. of MSB of mem. size
4E0C BE          CP      (HL)          Is DCB at end of memory?
4E0D 2802        JR      Z,4E11H       Error if yes
4E0F 3054        JR      NC,4E65H      Error if DCB addr. above mem. size
4E11 1A          LD      A,(DE)        Load 1st byte of DCB
4E12 CB7F        BIT     07H,A         Is file open
4E14 284F        JR      Z,4E65H       Error if not
```

After these tests have been performed, the DCB address is copied from the DE register to the IX register by the intructions at **4E16 - 4E17**. Then three more tests are performed. The first two test the buffer address, insuring it is RAM resident and below the end of memory. The third test verifies that the drive number is between zero and three. The following code is used for these tests:

Figure 6.2 *SYS3 Code From 4E16 to 4E2E*

```
4E16 D5          PUSH    DE            DCB addr. to stack
4E17 DDE1        POP     IX            Then load it into IX
4E19 DD7E04      LD      A,(IX+04H)    A = MSB of buffer addr.
4E1C FE40        CP      40H           Is buffer in system area
4E1E 3845        JR      C,4E65H       Error if yes
4E20 BE          CP      (HL)          Is buffer start at end of memory
4E21 2802        JR      Z,4E25H       Error if so, or if buffer
4E23 3040        JR      NC,4E65H      start above end of memory
4E25 DD7E06      LD      A,(IX+06H)    Load drive number
4E28 FE00        CP      00H           Test for illegal drive number
4E2A 3839        JR      C,4E65H       Error if drive out of range
4E2C FE04        CP      04H           Compare with max. drive number
4E2E 3035        JR      NC,4E65H      Error if drive out of range
```

After the buffer address has been validated, furthur tests on the DCB parameters are performed. Beginning at **4E30**, the unused bits of the HIT index are tested to insure they are zero. Next several tests are applied to the track number in the first GAP. The first test checks for operations against track 0, granule 0. This appears to be a precaution against the accidental KILL'ing of the boot loader; however, since it is applied to the first GAP only, the test is incomplete.

The second test compares the track number against the maximum number of tracks on the drive. This is an indirect system limitation inasmuch as it assumes all drives have the same number of tracks. As with the first test, it is only applied to the first GAP.

Following these tests, the SYS3 request code option is examined, and control is passed to the requested function. The SYS3 request code options are:

Figure 6.3 *Request Code Options*

REQUEST CODE OPTIONS	FUNCTION
10	CLOSE
20	KILL

The following code is used for the DCB tests described above. The code at **4E65** is the exit path taken if any errors are detected with the DCB paramters, the DCB address or the buffer address.

Figure 6.4 *SYS3 Code From 4E30 to 4E6C*

```
4E30 DD7E07      LD      A,(IX+07H)    Load HIT index
4E33 CB67        BIT     04H,A         Test unused bit (should be 0)
4E35 202E        JR      NZ,4E65H      Error if on (HIT) = 10)
4E37 CB5F        BIT     03H,A         Test unused bit (should be 0)
4E39 202A        JR      NZ,4E65H      Error if on (HIT) = 08)
4E3B DD7E0E      LD      A,(IX+0EH)    Load track number 1st GAP
4E3E FE00        CP      00H           Are any tracks assigned
4E40 200A        JR      NZ,4E4CH      Jump if yes
4E42 DD7E0F      LD      A,(IX+0FH)    No, fetch gran count (may be track 0)
4E45 FE20        CP      20H           If track 0, must be 2nd granule
4E47 381C        JR      C,4E65H       Error if 1st granule (track 0,
                                       sector 0 reserved for system use)
4E49 DD7E0E      LD      A,(IX+0EH)    Reload track number 1st GAP
4E4C FEFF        CP      0FFH          Test for no tracks assigned
4E4E 2804        JR      Z,4E54H       Jump if none assigned
4E50 FE23        CP      23H           Greater than max. no. of tracks
4E52 3011        JR      NC,4E65H      Error if track greater than 34
*
*     Parameter validation complete. Test for
*     CLOSE or KILL option.
*
4E54 F1          POP     AF            Restore request code
```

Listing Continued . . .

. . . Continued Listing

```
4E55 E1          POP     HL          Restore buffer address
4E56 DDE1        POP     IX          Restore DCB address
4E58 E670        AND     70H         Isolate option code
4E5A FE10        CP      10H         Test for CLOSE request
4E5C CA6D4E      JP      Z,4E6DH     Jump if CLOSE call
4E5F FE20        CP      20H         Test for KILL request
4E61 CAD04F      JP      Z,4FD0H     Jump if KILL call
4E64 C9          RET                 Neither, ignore request
4E65 F1          POP     AF          Various errors. Restore req. code
4E66 E1          POP     HL          Buffer addr.
4E67 DDE1        POP     IX          And DCB addr.
4E69 3E26        LD      A,26H       Error code for file not opened
4E6B B7          OR      A           Set status flags
4E6C C9          RET                 Return with error status
```

6.1 Close Processing

Code for CLOSE processing begins at **4E6D** and ends at **4FCF**. Except for calls to subroutines in the nucleus, the code is in-line and has very few loops.

Functions to be performed during a CLOSE operation are summarized below.

1. Flush the sector buffer to disk if the file was opened for logical I/O operations.

2. Read the primary directory entry and update the EOF pointers to point to the current end of file, as indicated by the DCB.

3. Compare the number of granules in the directory for the file to the number of granules in the DCB. If the file has grown smaller (the granule count in the DCB is less than the granule count in the directory sector), release the granules assigned to unused GAP's.

4. Reconstruct the filename, extension, and drive number in the DCB.

Step one uses a subroutine in the nucleus to flush the sector buffer. The following calling sequence is used:

Figure 6.5 *SYS3 Code From 4E6D to 4E74*

```
*
*                ---- CLOSE Processing ----
*
4E6D CD9248      CALL    4892H       Save caller's registers
4E70 C0          RET     NZ          Error exit if file not open
4E71 CD7848      CALL    4878H       Flush sector buffer if necessary
4E74 C0          RET                 Exit if error while purging buffer
```

The second step requires slighly more code than step one, some of which is executed conditionally. There are two major sections of code in step two.

The first reads the directory entry into the sector buffer at **4200** and compares the EOF pointer values in the directory to those in the DCB. Since the DCB may be considered to be the most recently modified data structure, it is assumed to be correct. If there is any difference between the directory and DCB pointers, those from the DCB are copied to the directory, and the updated directory sector is rewritten to disk. The following code is used:

Figure 6.6 *SYS3 Code From 4E75 to 4EB5*

```
4E75 DD4607     LD      B,(IX+07H)   B = HIT Index (Dir sector no.)
4E78 DD4E06     LD      C,(IX+06H)   C = Unit
4E7B CDC14A     CALL    4AC1H        Read directory entry into 4D00.
                                     HL = addr. of entry
4E7E C0         RET     NZ           Exit if error during read
4E7F 3E03       LD      A,03H        Offset to EOF byte
4E81 85         ADD     A,L          Add offset to addr. of entry
4E82 6F         LD      L,A          Reform addr.
4E83 E5         PUSH    HL           Save addr. of EOF byte
4E84 DD7E08     LD      A,(IX+08H)   A = EOF offset from DCB
4E87 BE         CP      (HL)         DCB EOF = Dir entry EOF?
4E88 2014       JR      NZ,4E9EH     No, go update dir entry EOF
4E8A 3E11       LD      A,11H        Inc to EOF sector no. in directory
4E8C 85         ADD     A,L          Add to current directory address
4E8D 6F         LD      L,A          Reform directory address
4E8E DD7E0C     LD      A,(IX+0CH)   LSB of EOF sector from DCB
4E91 BE         CP      (HL)         Compare to LSB of EOF sector
                                     from directory
4E92 200A       JR      NZ,4E9EH     If unequal, update dir EOF sector #
4E94 2C         INC     L            Bump to MSB of EOF sector # in dir
4E95 DD7E0D     LD      A,(IX+0DH)   MSB of EOF sector from DCB
4E98 BE         CP      (HL)         Compare to MSB of EOF sector in dir
4E99 2003       JR      NZ,4E9EH     If unequal, update dir EOF sector #
4E9B F1         POP     AF           Clear stack
4E9C 1818       JR      4EB6H        Go compute total no. of grans
4E9E E1         POP     HL           Restore base address of
                                     directory entry + 3
4E9F DD7E08     LD      A,(IX+08H)   Load EOF byte offset from DCB
4EA2 77         LD      (HL),A       And copy to directory
4EA3 3E11       LD      A,11H        Offset to EOF sector no. in dir
4EA5 85         ADD     A,L          Add to current directory address
4EA6 6F         LD      L,A          And reform address in HL
4EA7 DD7E0C     LD      A,(IX+0CH)   Fetch LSB of ending sector
                                     number from DCB
4EAA 77         LD      (HL),A       Move it to directory entry
4EAB 2C         INC     L            Bump to addr. of MSB of ending
                                     sector in directory
```

Listing Continued . . .

. . . Continued Listing

```
4EAC DD7E0D      LD      A,(IX+0DH)    Fetch MSB of ending sector no.
4EAF 77          LD      (HL),A        And copy to directory
4EB0 E5          PUSH    HL            Save current directory address
4EB1 CDD64A      CALL    4AD6H         Write uupdated directory entry
4EB4 E1          POP     HL            Restore dir. addr.
4EB5 C0          RET     NZ            Exit if error during write
```

The second major portion of code for step two computes the total number of granules assigned to the file by summing the graunlue counts from the GAP's in the directory. This code can get a little involved since it must handle overflow directory entries (entries which hold additional GAP's). The code for this operation begins at **4EB6** by positioning the address in the HL register to the first GAP, and clearing the accumulator used for the granule totals.

A loop from **4EBA - 4ED7** scans the GAP's, accumulating the total granule count. The loop terminates when an end of GAP indicator (**FF**) is found.

If an overflow GAP indicator (**FE**) is encountered, the overflow directory entry is read by calling a nucleus subroutine. After the overflow entry has been read, the main loop is re-entered, and the scanning continues. Following is the code used for this step:

Figure 6.7 *SYS3 Code From 4EB6 to 4ED7*

```
4EB6 2C          INC     L             Bump to track number for 1st GAP
4EB7 110000      LD      DE,0000H      Zero storage for total granules
4EBA 7E          LD      A,(HL)        Fetch starting track no. from a GAP
4EBB 2C          INC     L             Bump pointer to granule count
4EBC FEFE        CP      0FEH          Test for overflow or end of GAPS
4EBE 300C        JR      NC,4ECCH      Jump if overflow or end of GAPS
4EC0 7E          LD      A,(HL)        A=No. of granules
4EC1 2C          INC     L             Bump to 1st byte of next GAP
4EC2 E61F        AND     1FH           Isolate count of granules
4EC4 3C          INC     A             Gives true value
4EC5 83          ADD     A,E           Accumulate total no. of grans in DE
4EC6 5F          LD      E,A           LSB of granule total
4EC7 30F1        JR      NC,4EBAH      Jump if LSB not over 255 yet
4EC9 14          INC     D             If it is, bump MSB
4ECA 18EE        JR      4EBAH         Loop till all granules summed
4ECC 200B        JR      NZ,4ED9H      Jump if end of GAPS reached
4ECE 46          LD      B,(HL)        Load HIT index for dir overflow
4ECF CDC14A      CALL    4AC1H         Read overflow directory entry
4ED2 C0          RET     NZ            Exit if error during read
4ED3 7D          LD      A,L           Get LSB of directory address
4ED4 C616        ADD     A,16H         Then add 16 to reach GAP address
4ED6 6F          LD      L,A           Reform addr. of GAPS in HL
4ED7 18E1        JR      4EBAH         Scan till end of GAPS reached
```

Listing Continued . . .

. . . Continued Listing

```
4EC9 14          INC     D              It has, bump MSB
4ECA 18EE        JR      4EBAH          Loop till all granules summed
4ECC 200B        JR      NZ,4ED9H       Jump if end of GAPS reached
4ECE 46          LD      B,(HL)         Load HIT index for overflow
                                        directory entry
4ECF CDC14A      CALL    4AC1H          Read overflow directory
4ED2 C0          RET     NZ             Exit if error during read
4ED3 7D          LD      A,L            Get LSB of directory address
4ED4 C616        ADD     A,16H          Then add 16 to get to the GAP
                                        address
4ED6 6F          LD      L,A            Reform addr. of GAPS in HL
4ED7 18E1        JR      4EBAH          And continue scan till end of
                                        GAPS reached
```

The third step represents the largest amount of code for CLOSE processing. As with the code in step two, portions of this code are executed conditionally.

The code in step three can be divided into two sections. The first computes the total number of granules currently assigned to the file by dividing the number of records in the file (fetched from the DCB) by five (5 = the number of sectors in a granule). This granule count is then compared to the granule count computed in step two using the directory GAP's. If the DCB granule count is smaller than the directory granule count, the directory entry for the file contains old GAP's. The old GAP's will be released in section two of step three. Following is the code used for the first section of step three:

Figure 6.8 *SYS3 Code From 4ED9 to 4EEF*

```
*
*
*
*     Test for change in file size using following method:
*     Directory file size (granule total from directory
*     GAP's) (minus) true file size (granule total from
*     ending sector in *DCB/5)
*
*     yields:
*
*     (Minus, if file has increased)
*     (zero, if file size is unchanged)
*     (positive, if file has decreased)
*
*
*
4ED9 E5          PUSH    HL             Save current addr in dir. End of GAPS
4EDA DD6E0C      LD      L,(IX+0CH)     L = LSB of number of records
4EDD DD660D      LD      H,(IX+0DH)     H = MSB of number of records
4EE0 3E05        LD      A,05H          A = Divisor
```

Listing Continued . . .

. . . Continued Listing

```
4EE2 CD844B     CALL    4B84H       Modulo divide HL by 5
4EE5 23         INC     HL          Quotient + 1
                                    (no. of grans = true size)
4EE6 AF         XOR     A           Clear carry
4EE7 EB         EX      DE,HL       HL = # of granules in directory.
                                    DE = true file size
4EE8 ED52       SBC     HL,DE       Compare true file size (by record
                                    count in DCB) to directory file
4EEA EB         EX      DE,HL       size (by GAP's). Save diff. in DE
4EEB E1         POP     HL          Restore current addr. in
                                    directory. End of GAPS
4EEC CA7B4F     JP      Z,4F7BH     If dir gran count is equal to or
4EEF DA7B4F     JP      C,4F7BH     greater, don't update directory
```

Section two of step three is the loop bounded by **4F01** through **4F6D**. Prior to entering this section of code, the GAT sector is read into the system buffer at **4200**, and the HIT sector is read into a local buffer at **5100**.

The GAT sector is read because as the old GAPs are released from the directory entry, the last granule associated with the GAP released has its entry in the GAT cleared. The HIT sector read is necessary because as the GAP entries in a directory entry are released, it is possible that an entire directory entry will be released (this should only happen with overflow entries).

Careful analysis of this code reveals several serious flaws in its implementation. First, the mechanics are wrong. The code from **4F07** - **4F29** clears the bit in the GAT for the GAP owning the last granule supposedly not in use. One problem with this code is that the granule could have been re-assigned to another file, for example, and therefore should not be released. A second problem with this code is that it does not account for multiple granules. Consider a GAP for track 20 with 4 granules assigned. That means track 20, granules 0 and 1, and track 21, granules 0 and 1, have been assigned consecutively. This code would clear the availability flag for track 21, granule 1, only. Clearly the intent was to make available the granules for tracks 20 and 21.

Considering the bug in the logic regarding re-assigned granules, this subsequent bug might be considered a stroke of luck. After all, if the code worked correctly, four re-assigned granules might incorrectly be flagged as available instead of one.

A second problem connected with the incorrect handling of the granule count on old GAP's affects the loop control. The code in the first section computed the difference in the number of granules between the directory and the DCB and left it in the DE register. After processing one GAP (incorrectly) the E register is decremented by one. It should have been decremented by the number of granules in the old GAP passed in the directory. This can lead to exhausting the GAP's prematurely in a directory entry.

One saving grace in this code is that it is almost impossible to generate the conditions which will cause it to be executed. To cause this code to be executed one must, via an assembly language program, modify the record count in the DCB. Since there is no good reason for doing this, the chances of encountering this bug are slim.

Following is the code for the second section of step three.

Figure 6.9 *SYS3 Code From 4EF2 to 4F7A*

```
4EF2 2D          DEC     L           Backspace to granule count
4EF3 2D          DEC     L           For last GAP
4EF4 E5          PUSH    HL          Save addr of last good GAP in dir
4EF5 D5          PUSH    DE          Save granule difference
4EF6 CDF04A      CALL    4AF0H       Read GAT sector into 4D00
4EF9 2003        JR      NZ,4EFEH    Jump if error during read. Skip
                                     read into 5100
4EFB CD6850      CALL    5068H       Read HIT sector into 5100
4EFE D1          POP     DE          Restore granule difference
4EFF E1          POP     HL          Restore addr of last GAP in dir
4F00 C0          RET     NZ          Exit if GAT read error either time
4F01 7B          LD      A,E         Load LSB of granule difference
4F02 FE00        CP      00H         Test if done
4F04 2869        JR      Z,4F6FH     Jump if done
4F06 D5          PUSH    DE          Save current granule difference
4F07 7E          LD      A,(HL)      Fetch granule count from last
                                     GAP in directory
4F08 E6E0        AND     0E0H        Isolate starting gran no. (bit 5)
4F0A 07          RLCA                Shift into low order of A register
4F0B 07          RLCA                Gives 0 if file starts on sector 0
4F0C 07          RLCA                of first track, or 1 if file begins
4F0D 5F          LD      E,A         on sector 5 of first track
4F0E 7E          LD      A,(HL)      Refetch gran count from last GAP
4F0F E61F        AND     1FH         Isolate no. of assigned grans
4F11 83          ADD     A,E         Add offset for 1st gran, save in E
4F12 5F          LD      E,A         Adds 0 if starting sect # is 0
                                     Adds 1 if starting sect # is 5
4F13 E6FE        AND     0FEH        Clear bit 0, so division will
                                     not affect carry
4F15 0F          RRCA                Divide granule count by 2
4F16 32224F      LD      (4F22H),A   And save number of tracks
4F19 7B          LD      A,E         Restore original gran count & offset
```

Listing Continued . . .

. . . Continued Listing

```
4F1A E601       AND     01H             Isolate Bit 0
*
*  Construct mask for clearing bit in GAT for last gran assigned
*  Use following rules:
*
*         ----------------------------------------
*          # of sectors in    sector      offset
*          granule count        0           1
*         ----------------------------------------
*           EVEN                FE          FD
*         ----------------------------------------
*           ODD                 FD          FE
*         ----------------------------------------
*
4F1C 3C         INC     A               Then add 1 and compliment
4F1D 2F         CPL                     to generate a mark for clearing
4F1E F5         PUSH    AF              the last assigned granule in
                                        the GAT. Save mask
4F1F 2B         DEC     HL              Backspace GAP pointer to track no.
4F20 7E         LD      A,(HL)          Fetch starting track number for GAP
4F21 C600       ADD     A,00H           Add no. of consecutively assigned
                                        tracks. Gives ending track #
4F23 EB         EX      DE,HL           Save directory GAP addr. in DE
4F24 264D       LD      H,4DH           Form address of GAT entry for
4F26 6F         LD      L,A             ending track in HL
4F27 F1         POP     AF              Restore mask used to clear last
                                        granule assigned
4F28 A6         AND     (HL)            Clear bit for last gran assigned
4F29 77         LD      (HL),A          And restore byte with cleared
                                        bit to GAT buffer
*
*    Look for GAPS which have no granules assigned. When one
*    is found, set both bytes of the GAP entry to FF. If all
*    GAP's in an entry are released (set to FF's), release the
*    directory entry and the HIT index in the HIT sector for
*    the directory entry.
*
4F2A EB         EX      DE,HL           Restore directory GAP addr to HL
4F2B 23         INC     HL              Skip forward to number of
                                        granules assigned this GAP
4F2C 35         DEC     (HL)            Prepare to test for no grans assigned
4F2D 7E         LD      A,(HL)          Fetch # granules assigned - 2
4F2E 3C         INC     A               Add 1. Yields  zero if none assigned
4F2F E61F       AND     1FH             Isolate true gran count for GAP
4F31 D1         POP     DE              Restore difference in granule
                                        count between DCB & dir.
4F32 2038       JR      NZ,4F6CH        Jump if at least one gran assigned
4F34 36FF       LD      (HL),0FFH       No granules assigned to this GAP
4F36 2B         DEC     HL              Initialize GAP to FF
4F37 36FF       LD      (HL),0FFH       Including track number
4F39 2B         DEC     HL              Backspace to gran count for last GAP
4F3A 7D         LD      A,L             Then isolate GAP index
4F3B E61F       AND     1FH             And test for start of GAP
4F3D FE15       CP      15H             entries in directory. If not
4F3F 202B       JR      NZ,4F6CH        start of GAP area - Go
```

Listing Continued . . .

. . . Continued Listing

```
4F41 AD          XOR     L           Form LSB of dir entry addr in A
4F42 6F          LD      L,A         Then reform start of directory
                                     entry address in HL
4F43 7E          LD      A,(HL)      Fetch access and dir type flags
4F44 3600        LD      (HL),00H    Flag directory entry as available
4F46 23          INC     HL          Bump pointer to overflow byte
                                     pointer in directory
4F47 56          LD      D,(HL)      Fetch possible overflow pointer
4F48 2651        LD      H,51H       Origin of HIT buffer to H
4F4A 68          LD      L,B         HIT index for file to HL.
                                     HL = addr.of HIT index
4F4B 3600        LD      (HL),00H    Clear HIT entry for dir entry
4F4D CB7F        BIT     07H,A       Test type of dir entry being cleared
4F4F 281E        JR      Z,4F6FH     Jump if primary directory entry
4F51 D5          PUSH    DE          Save remainder of gran count diff
4F52 CDD64A      CALL    4AD6H       Write released directory entry
4F55 D1          POP     DE          Restore granule count difference
4F56 C0          RET     NZ          Exit if error during sector rewrite
4F57 78          LD      A,B         Fetch HIT index and compare
4F58 AA          XOR     D           Sector no. portion of it with
4F59 E607        AND     07H         the sector no. from overflow pointer
4F5B 42          LD      B,D         Save sector number for next
                                     overflow pointer
4F5C C4C14A      CALL    NZ,4AC1H    Read overflow directory entry
                                     (if there is one)
4F5F C0          RET     NZ          Exit if error during read
4F60 78          LD      A,B         A = entry addr/sector no. -2
4F61 E6E0        AND     0E0H        Isolate directory entry address
4F63 C61F        ADD     A,1FH       Skip to end of GAP's in overflow
                                     directory entry
4F65 6F          LD      L,A         Form ending dir addr in HL
4F66 36FF        LD      (HL),0FFH   Signal end of GAP's
4F68 2B          DEC     HL          and backspace to starting track no.
4F69 36FF        LD      (HL),0FFH   Initialize it on end of GAP's
4F6B 2B          DEC     HL          Backspace to gran count for next GAP
4F6C 1D          DEC     E           Reduce difference in gran count
4F6D 1892        JR      4F01H       and go test if done
*
*     Write updated directory entry. Reconstruct DCB
*
4F6F CDD64A      CALL    4AD6H       Write updated directory entry
4F72 C0          RET     NZ          Exit if error during write
4F73 CD7750      CALL    5077H       Maybe write updated HIT sector
4F76 C0          RET     NZ          Exit if error during HIT rewrite
4F77 CD034B      CALL    4B03H       Write GAT sector
4F7A C0          RET     NZ          Exit if error during write
```

The code for the fourth step is almost as straightforward as that for the first step. It reconstructs the filename, extensions and drive number and stores them in the DCB. The only wrinkle in this code is at the beginning, where the HIT index from the DCB is compared to the current HIT index in the B register. This test is necessary to insure the primary directory entry occupies the system buffer at **4200**. Depending on the actions taken during step three, another directory could have been read into the system buffer.

The remainder of the code for step four copies the filename and extension to the DCB. The drive number is also converted to ASCII and stored in the DCB. Following these operations, control returns to the caller of SYS3.

The following code is used for the fourth step:

Figure 6.10 *SYS3 Code From 4F7B to 4FCF*

```
4F7B DD7E07      LD      A,(IX+07H)    Load HIT index
4F7E A8          XOR     B             Compare with last HIT index
4F7F E61F        AND     1FH           Isolate any possible difference
4F81 C4C14A      CALL    NZ,4AC1H      They were different, read
                                       directory entry
4F84 C0          RET     NZ            Exit if error during read
4F85 DD7E06      LD      A,(IX+06H)    Load drive number
4F88 E603        AND     03H           Force a value between 0 and 3
4F8A F630        OR      30H           Reconstruct as ASCII value
4F8C 32C84F      LD      (4FC8H),A     to be restored into DCB
4F8F 2642        LD      H,42H         Form address of file name in dir
4F91 DD7E07      LD      A,(IX+07H)    Get the HIT index
4F94 E6E0        AND     0E0H          Isolate offset to file entry
4F96 F605        OR      05H           Add a 5 and move to L
4F98 6F          LD      L,A           Gives addr. of file name
                                       in directory entry
4F99 E5          PUSH    HL            Save addr. of file name in dir
4F9A DDE5        PUSH    IX            Start of DCB address to DE
4F9C D1          POP     DE            Prepare to rebuild DCB for OPEN call
4F9D 0608        LD      B,08H         Max. no. of chars in file name
4F9F 7E          LD      A,(HL)        Get a char. from file name
4FA0 FE20        CP      20H           Is it a blank
4FA2 2805        JR      Z,4FA9H       Jump if yes. File name copied
4FA4 12          LD      (DE),A        Move  char from directory to DCB
4FA5 23          INC     HL            Bump directory addr.
4FA6 13          INC     DE            Bump DCB addr.
4FA7 10F6        DJNZ    4F9FH         Loop till name copied to DCB
4FA9 E1          POP     HL            Restore addr of name in directory
4FAA 7D          LD      A,L           Then add 8 to it to
4FAB C608        ADD     A,08H         Get addr of extension in directory
4FAD 6F          LD      L,A           HL = Addr. of file ext.
4FAE 7E          LD      A,(HL)        Fetch a character of extension
4FAF FE20        CP      20H           Is it a blank
4FB1 2810        JR      Z,4FC3H       Jump if yes, no extension
4FB3 3E2F        LD      A,2FH         Else, save a "/"
```

Listing Continued . . .

. . . Continued Listing

```
4FB5 12          LD      (DE),A      After file name in DCB
4FB6 13          INC     DE          Bump to pos. of 1st ext. char in DCB
4FB7 0603        LD      B,03H       Max. no. of chars in extension
4FB9 7E          LD      A,(HL)      Fetch a character of extension
4FBA FE20        CP      20H         Is it a blank
4FBC 2805        JR      Z,4FC3H     Go if so. End of ext. found
4FBE 12          LD      (DE),A      Else save extension character
4FBF 23          INC     HL          Bump directory address
4FC0 13          INC     DE          Bump DCB address
4FC1 10F6        DJNZ    4FB9H       Loop till ext. copied to DCB
4FC3 3E3A        LD      A,3AH       Next, save an ASCII :
4FC5 12          LD      (DE),A      in DCB, then
4FC6 13          INC     DE          Bump to next addr.
4FC7 3E00        LD      A,00H       Load ASCII drive number
4FC9 12          LD      (DE),A      And save it in DCB
4FCA 13          INC     DE          Bump to first char after drive no.
4FCB 3E03        LD      A,03H       And store a
4FCD 12          LD      (DE),A      terminator
4FCE AF          XOR     A           Signal good status
4FCF C9          RET                 Return to caller
```

6.2 Kill Processing

Code for KILL processing begins at **4FD0** and ends at **5038**. The processing is remarkably simple. The primary directory entry is read into the system buffer at **4200**, the GAT is read into the system buffer at **4D00** and the HIT sector is read into the local buffer at **5100**. Next, the GAP's in the direcctory are processed one by one; the bit in the GAT associated with each granule being processed is reset (flagged as available). When all GAP's have been processed, the directory entry is zeroed, and the HIT index in the HIT sector is cleared. Finally, all three sectors (the directory, GAT, and HIT), are rewritten to disk.

These operations can be broken into the following steps:

1. Test for KILL permission; exit with error if not true. Read the GAT, HIT, and directory sectors.

2. Process all GAP's in the directory one at a time. For each GAP, call a de-allocate subroutine which releases all granules owned by the GAP. If overflow GAP pointers are encountered, read the overflow entries and process the GAP's.

3. Zero the directory entry and HIT index, and write the updated sectors to disk.

Step one begins at **4FD0** and ends at **4FF2**. The code is quite straightforward. After verifying the permission flags (**4FD4** - **4FBD**), the required disk sectors are read via calls to the nucleus.

Following is the code used for step one.

Figure 6.11 *SYS3 Code From 4FD0 to 4FF2*

```
*
*                  ---- KILL Processing ----
*
4FDØ CD9248      CALL    4892H          Save caller's registers
4FD3 CØ          RET     NZ             Error exit if file not open.
4FD4 DD7E01      LD      A,(IX+01H)     Get permission flags
4FD7 E607        AND     07H            Isolate them
4FD9 FE02        CP      02H            Test if KILL pe`mission
4FDB 3804        JR      C,4FElH        Jump if KILL permission. Else
4FDD 3E25        LD      A,25H         rret illegal file access code
4FDF B7          OR      A              Set error status flags
4FE0 C9          RET                    And return to caller
4FEl DD4E06      LD      C,(IX+06H)     C = Unit no.
4FE4 DD4607      LD      B,(IX+07H)     B = HIT index number
4FE7 CD6850      CALL    5068H          Read HIT sector into 5100
4FEA CØ          RET     NZ             Exit if error during read
4FEB CDF04A      CALL    4AF0H          Read GAT into 4D00
4FEE CØ          RET     NZ             Exit if error
4FEF CDC14A      CALL    4AClH          Read directory entry into 4200.
                                        HL = addr. of entry
4FF2 CØ          RET     NZ             Exit if error during read
```

The second step is composed of three parts. The first part is a loop, which begins at **4FF3** and ends at **5003**. The loop processes all GAP's in a directory entry by calling a subroutine at **5039** to release all granules assigned to a particular GAP. Folowing is the code for part one of step two:

Figure 6.12 *SYS3 Code From 4FF3 to 5003*

```
4FF3 3E16        LD      A,16H          Offset to start of GAP's
4FF5 85          ADD     A,L            Combine with dir entry address
4FF6 6F          LD      L,A            Form GAP addr for dir entry in HL
4FF7 5E          LD      E,(HL)         Fetch track address from GAP
4FF8 2C          INC     L              Bump pointer to granule count
4FF9 56          LD      D,(HL)         Fetch no. of assigned grans
4FFA 7B          LD      A,E            Prepare to test for end of GAP's
4FFB FEFE        CP      0FEH           End of GAP's or overflow pointer?
4FFD 3006        JR      NC,5005H       Jump if yes
4FFF 2C          INC     L              Else bump to next GAP entry
5000 CD3950      CALL    5039H          Deallocate GAP
5003 18F2        JR      4FF7H          Loop till end of GAP's reached,
                                        or overflow pointer found
```

The second part of step two consists of the subroutines called to de-allocate a GAP. The main subroutine which is called with a GAP entry as a parameter occuppies **5039 - 505A**. It calls a subroutine at **505B@T - 5067** on a granule basis to flag the granule as available in the GAT.

The method used by both subroutines is instructive. In the **5039** subroutine, a loop from **5047 - 5056** processes each GAP. Imbedded in this loop is a call to the other subroutine starting at **505B**.

The subroutine at **505B** must be called with the bit number for the GAT entry to be reset (set to zero), in the A register. Since the bit numbers associated with granules can only be zero or one, a counter for the bit being cleared must be maintained. The code from **503F** to **5046** initializes this counter to the proper starting bit position.

The subroutine at **505B** uses the bit number passed in the A register to construct a RES X,B instruction, where X is the bit position (0 or 1 for GAT entries).

Following is the code used for the de-allocation portion of step two:

Figure 6.13 *SYS3 Code From 5039 to 5067*

```
*
*                    GAP Deallocation
*
5039 E5          PUSH    HL          Save current dir address
GAP deallocation
503A C5          PUSH    BC          Save HIT index/unit number
503B 6B          LD      L,E         Track number to L. Becomes index
into GAT
503C 264D        LD      H,4DH       Form addr of starting track no. in HL
                                     number in HL
503E 7A          LD      A,D         Get number of consecutively
assigned grans
503F E61F        AND     1FH         Isolate granule count -1
5041 4F          LD      C,A         Save in C
5042 0C          INC     C           And compute true granule count
5043 AA          XOR     D           Isolate starting gran no. flag
                                     (0 = sect. 0, 1 = sect. 5)
5044 07          RLCA                and reposition
5045 07          RLCA                into
5046 07          RLCA                lower bit of A-reg.
5047 F5          PUSH    AF          Save it as the initial bit
                                     counter for each GAT entry
5048 46          LD      B,(HL)      Fetch current GAT entry for
                                     current track number
5049 CD5B50      CALL    505BH       Release a granule (make it
                                     available) for current GAT entry
504C 70          LD      (HL),B      Restore updated GAT entry
504D F1          POP     AF          Get bit counter for current GAT entry
504E 3C          INC     A           Bump it and
```

Listing Continued . . .

. . . Continued Listing

```
504F FE02        CP      02H          test if Max. # granules released
                                      for this entry
5051 2002        JR      NZ,5055H     Jump if more granules to be
                                      released for this GAT entry
5053 AF          XOR     A            Else, reset bit counter to zero
5054 2C          INC     L            Bump to GAT entry for next track
5055 0D          DEC     C            Count 1 granule released
5056 20EF        JR      NZ,5047H     Jump if more grans to be released
5058 C1          POP     BC           Restore HIT index, unit number
5059 E1          POP     HL           Restore current dir GAP address
505A C9          RET                  Return to caller
*
*        Clear bit number in B specified by A
*
505B E607        AND     07H          Isolate bit number to be reset.
                                      Will be 0 or 1
505D 07          RLCA                 and
505E 07          RLCA                 position for
505F 07          RLCA                 placement into RES instr.
5060 F680        OR      80H          Combine RES Op code with bit to clear
5062 326650      LD      (5066H),A    Save synthesized RES instr.
5065 CB80        RES     00H,B        Now clear bit for this granule
5067 C9          RET                  Return to caller
```

The third part of step two is executed on an exception basis. It is called when the GAP's in the directory entry have been exhausted and an overflow pointer encountered. This code must zero the current directory entry, clear the HIT index, rewrite both sectors and read the directory entry indicated by the overflow pointer. The section of code that zeros the buffers and rewrites them is common to step three. Following is the code for the third part of step two:

Figure 6.14 *SYS3 Code From 5005 to 5024*

```
5005 7D          LD      A,L          Get LSB of directory entry addr.
                                      in sector buffer
5006 E6E0        AND     0E0H         Isolate starting address of entry
5008 6F          LD      L,A          And form beginning address in HL
5009 3600        LD      (HL),00H     Clear access flags, flag entry
                                      as available
500B C5          PUSH    BC           Save HIT index/unit number
500C D5          PUSH    DE           Save GAP flags (FF or FE)
500D E5          PUSH    HL           Start of directory entry address
500E D1          POP     DE           To DE
500F 13          INC     DE           Bump to 2nd byte of directory
                                      entry address
5010 011F00      LD      BC,001FH     Number of bytes to move
5013 EDB0        LDIR                 Zero directory entry in sector
                                      buffer
5015 D1          POP     DE           Restore GAP terminator flag
5016 C1          POP     BC           Restore HIT index/unit number
```

Listing Continued . . .

. . . Continued Listing

```
5017 CDD64A     CALL    4AD6H       Write drectory sector with
                                    zeroed entry
501A C0         RET     NZ          Exit if error during write
501B 2651       LD      H,51H       Load MSB of HIT buffer address,
                                    then load HIT index
501D 68         LD      L,B         To form address of byte
                                    containing hash code for file
501E 3600       LD      (HL),00H    Remove hash code from HIT sector
                                    buffer
5020 42         LD      B,D         B = sector # of possible
                                    overflow entry
5021 7B         LD      A,E         Fetch GAP terminating flag
5022 FEFE       CP      0FEH        Test for overfow entry
5024 28C9       JR      Z,4FEFH     Jump if overflow entry. Go read
                                    and process
```

The third step is very brief. It unconditionally writes the updated GAT and HIT sector. It is assumed that when this code is reached, the updated directory has been rewritten. After writing the updated sectors, the caller's DCB is zeroed. Following is the code for step three:

Figure 6.15 *SYS3 Code From 5026 to 5038*

```
5026 CD034B     CALL    4B03H       Else, write updated GAT sector
                                    from buffer at 4D00
5029 C0         RET     NZ          Exit if error during write
502A CD7750     CALL    5077H       Write updated HIT sector from
                                    buffer at 5200
502D C0         RET     NZ          Exit if error during write
502E DDE5       PUSH    IX          Copy DCB address
5030 E1         POP     HL          to HL
5031 0620       LD      B,20H       Number of bytes to clear in DCB
5033 AF         XOR     A           Load a zero into A-reg.
5034 77         LD      (HL),A      Clear a byte in DCB
5035 23         INC     HL          Bump to next address in DCB
5036 10FC       DJNZ    5034H       Loop till DCB cleared
5038 C9         RET                 Return to KILL caller
```

notes

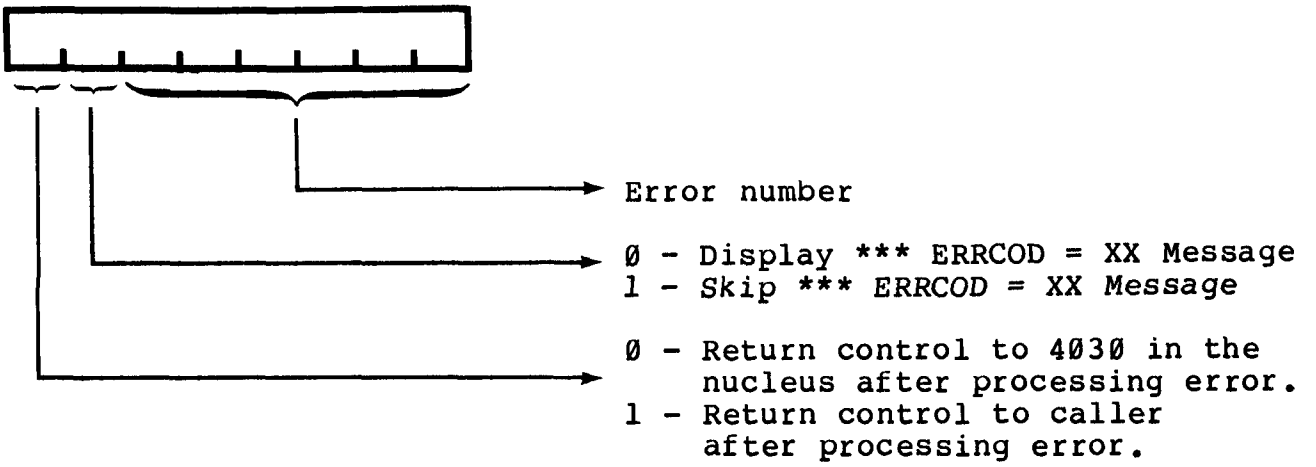
7

7.0 SYS4/SYS

SYS4/SYS is the error processing portion of TRSDOS. It is called from the nucleus when error conditions have been detected by the nucleus, any system overlay module or any utility program.

SYS4 begins at **4E00** and ends at **51F2**; the entry point is **4E00**. A single parameter containing an error code number and two flags is passed to SYS4 in the A register. This parameter has the following format:

Figure 7.1 *SYS4 Error Code Format*



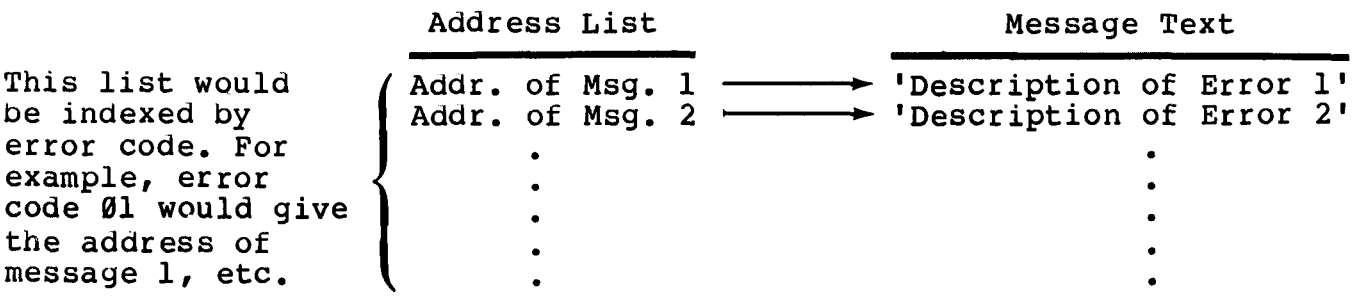
At first blush SYS4 appears to be more complicated than it really is. This is because it is mostly table-driven, whereas the other SYS modules work with a command line buffer, DCB, or well-known system data structures.

There are three tables in SYS4 that are used for processing every error code. These tables are used to translate the error code into an error message such as:

PARITY ERROR DURING READ

The typical way for programming an error message processor is to predefine every error message and construct a table containing the address of every message. The error code is then used as an index into the table of addresses to locate an error message corresponding to a particular error code. The data structures for this approach might appear as shown below:

Figure 7.2 *Sample Error Message Data Structures*



The major drawback to this method is the error message routine may become quite large because of the amount of space required for the ASCII text of the messages. If the message processor is a utility program, the messages can be read from a message file or disk, or overlays containing segments of error message can be loaded if the size of the processor becomes critical.

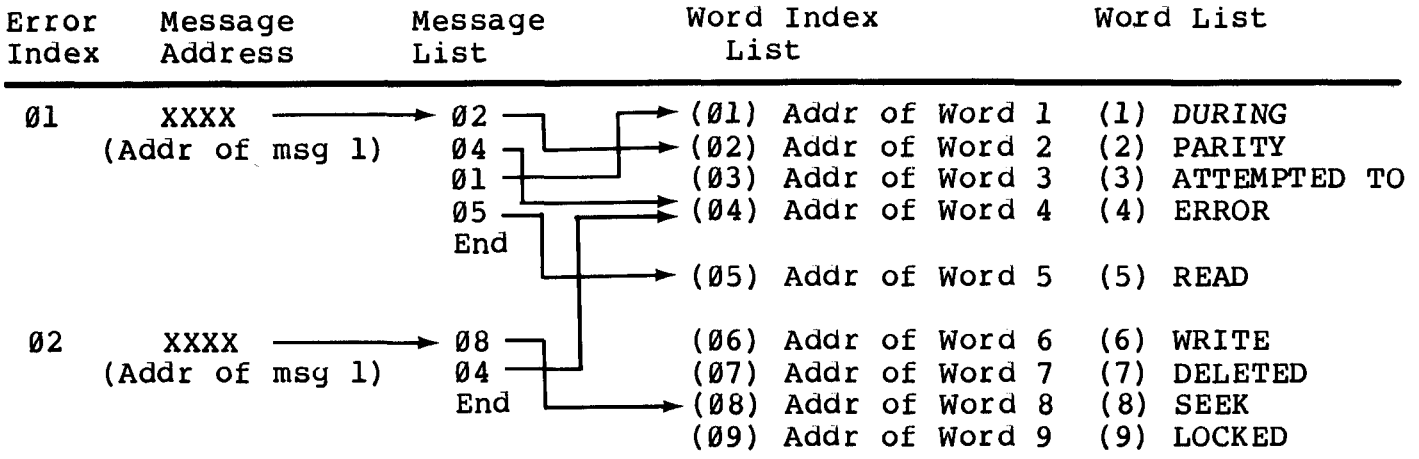
In TRSDOS, the message processor is an overlay, so the size of the message processor is a problem almost immediately (because of the large number of error messages). The messages could have been stored on a disk file and read as needed, but if the error concerns a disk problem, the message processor may not be able to read the error messages from the disk. Or, the system diskette containing the error file may not be mounted. Overlays for overlays are not practical for TRSDOS in this case because the overlay area is so small.

The approach taken with TRSDOS was to reduce the amount of space required for the error messages. This was accomplished by constructing a list containing every word used in any error message. Error messages were then built using indices into the list instead of words. This reduces the size of the error message because no words are repeated. Word list indices representing a word will be repeated, but they occupy only one byte. Thus, the messages become quite short.

The only limitation with this method, as implemented in TRSDOS, is the size of the word list, since the indices into the list are bytes, and only six bits of each byte are used as an index. The maximum number of words in a message is limited to sixty four.

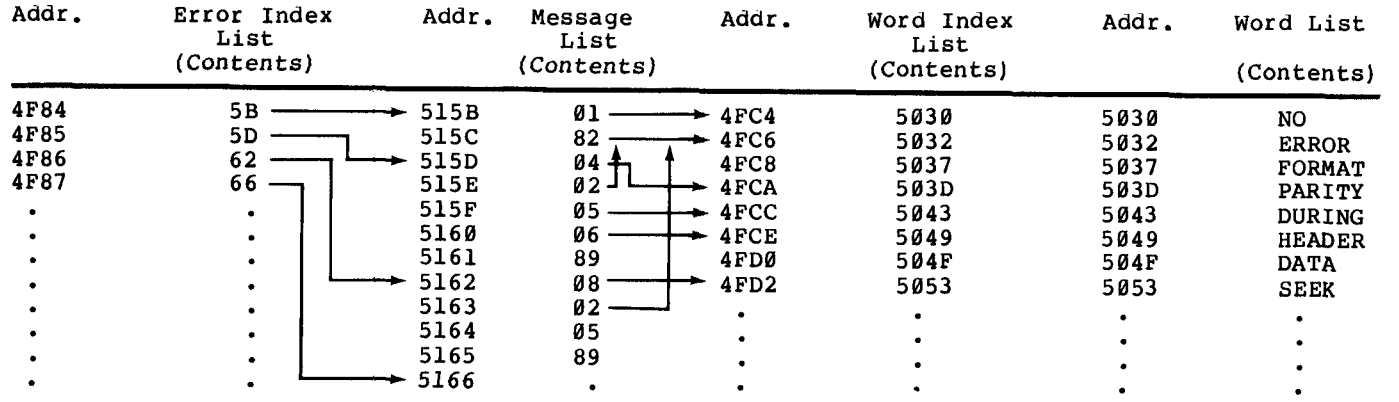
The following types of data structures were used to implement the word index method in TRSDOS. As an illustrative example, a limited word list is shown. The actual word list in SYS4 is much larger.

Figure 7.3 *Error Message Data Structure*



A description of the actual data structures used in SYS4 follows:

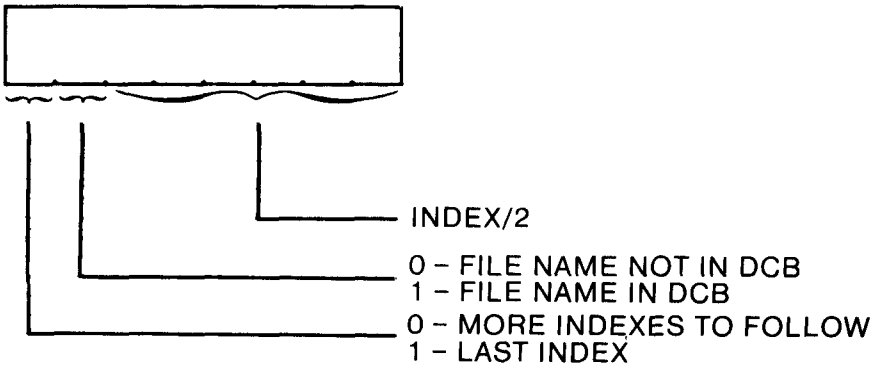
Figure 7.4 *SYS4 Data Structures*



Notice the further economies in the ERROR INDEX LIST and the MESSAGE LIST. The ERROR INDEX LIST has a one-byte pointer to the MESSAGE LIST, rather than a full 16-bit address. This requires the most significant byte of the address of every MESSAGE LIST address to be a known value. In this case, that value is 51. Thus, the contents of any byte in the ERROR INDEX LIST is the least significant byte of an address of the form 51XX, where the XX represents the contents of a particular byte of the ERROR LIST.

A second economy is achieved with the terminator byte in the MESSAGE LIST. The exact format of a terminator is given below:

Figure 7.5 *SYS4 Message Index Format*



Instead of having a separate terminator byte, a terminator flag is combined with the last index in the MESSAGE LIST.

7.1 SYS4 Processing

The entry conditions for SYS4 are different from those for any other system overlay module. The parameter byte containing the error number is passed as the

second push on the stack rather than in the A register as with the other modules. This difference is due to the fact that the parameter byte for SYS4 requires 8 bits, whereas the parameter field for all other system overlays is 4 bits.

The processing within SYS4 is straightforward. Three types of error messages may be generated depending upon the parameter passed and the error message associated with the error.

The simplest case occurs when Bit 6 of the parameter byte is non-zero. For that case, the message associated with the error number (Bit 0 - 5 of the parameter) is displayed. After displaying the message control returns to the caller, or **4030** in the nucleus, depending on Bit 7 of the parameter byte.

A second case occurs when Bit 6 of the parameter byte is zero. This case assumes the error was associated with a file. The file name and the sector buffer address associated with the file in error will be displayed following the main error message. For this case, the message

ERRCOD=XX

is displayed before the error message is dislayed. XX will be the ASCII equivalent of the error code passed in the parameter byte.

Following the error message, the file name and sector buffer address are displayed as:

<FILE=NNNNNNNN//EXT:D>
REFERENCED AT X'XXXX'

Prior to displaying these messages, the possibility of displaying the following message arises:

<DEVICE=*XX>

This message will appear automatically and supercede the <FILE=> display if both of the following conditions are true:

1. Bit 6 of the terminator index is non-zero. This is true for a limited number of messages.
2. The first byte of the file DCB contains an '*.'

If both conditions are true, the device code will be the value contained in the second and third bytes of the DCB.

A second set of conditions may cause the message to be displayed. Those conditions are:

1. Bit 6 of the terminator index must be zero.
2. The OPEN bit in the DCB must be zero. File not OPEN.

If both of these conditions are true, the device code will come from the seventh and eighth bytes of the DCB.

The exact circumstances which lead to either of these sets of conditions are unknown at this time. There is a possibility that some other use was intended for these conditions but never implemented.

When SYS4 is entered, the stack contains the return address of the SYS4 caller followed by the parameter byte. SYS4 begins by POP'ing the return address from the stack and saving it at **4E6E**. There are two reasons for this. First, following completion, SYS4 will return to the original caller or **4030** in the nucleus (depending on the option specified in the parameter byte). Second, the stack must be cleared to gain access to the parameter byte which precedes the return address on the stack.

Following those operations, the HL, DE and BC registers are saved, and bit 6 of the parameter byte is tested. If it is zero, the message:

ERRCODD=XX

is displayed. If the bit is non-zero, this message is skipped. The following code is used for the operations described up to this point:

Figure 7.6 *SYS4 Code From 4E00 to 4E27*

```
4EØØ E3          EX      (SP),HL        Return addr to HL. HL to stack
4EØ1 226E4E      LD      (4E6EH),HL     Return addr to exit instruction.
4EØ4 E1          POP     HL             Restore HL (clear stack)
4EØ5 F1          POP     AF             Restore error code
4EØ6 E5          PUSH    HL             Preserve HL
4EØ7 D5          PUSH    DE             DE
4EØ8 C5          PUSH    BC             BC
4EØ9 F5          PUSH    AF             And error code
4EØA 47          LD      B,A            Save original error code
4EØB E63F        AND     3FH            Clear upper two bits
4EØD 4F          LD      C,A            C = cleared error code
4EØE CB7Ø        BIT     Ø6H,B          Is ***ERRCOD=XX message flag set
4E1Ø ØEØØ        LD      B,ØØH          BC = error code (used as index
4E12 2Ø16        JR      NZ,4E2AH       Jump if ***ERRCOD msg flag off
*
*                    Hex to ASCII Conversion
*
4E14 1E3Ø        LD      E,3ØH          E = ASCII zero
4E16 D6ØA        SUB     ØAH            Subtract 1Ø from hex value
```

Listing Continued . . .

. . . Continued Listing

```
4E18 3803        JR      C,4E1DH      Jump if done with conversion
4E1A 1C          INC     E            Bump ASCII value
4E1B 18F9        JR      4E16H        Loop till ASCII equivalent computed
4E1D C63A        ADD     A,3AH        Make MSB of hex ASCII value positive
4E1F 57          LD      D,A          for ASCII error code in DE
4E20 ED53424F    LD      (4F42H),DE   Then save in ***ERRCOD=XX message
4E24 21364F      LD      HL,4F36H     Address of ***ERRCOD message
4E27 CD6744      CALL    4467H        Display msg.
```

Next, the error message associated with the error code is assembled and dislayed. The processing for this operation begins at **4E2A** and ends at **4E5A**.

At first, the code may appear confusisng. A full understanding of the lists from **4F84 - 4FC3**, **5l5B - 5lF2** and **4FC2 - 502F** will aid in comprehending this code:

Figure 7.7 *SYS4 Code From 4E2A to 4E5A*

```
4E2A 21844F      LD      HL,4F84H     Addr of error code index list
4E2D 09          ADD     HL,BC        Add error code to get address of
                                      phrase indices for error
4E2E 6E          LD      L,(HL)       L = LSB of phrase index address
                                      for error code
4E2F 2651        LD      H,51H        H = MSB of phrase index address
4E31 7E          LD      A,(HL)       Load a word index
4E32 E63F        AND     3FH          Clear terminator/unused lists
4E34 87          ADD     A,A          Index * 2 (list is addr by word
4E35 E5          PUSH    HL           Save current word index address
4E36 4F          LD      C,A          C = index * 2
4E37 0600        LD      B,00H        BC = index * 2
4E39 21C24F      LD      HL,4FC2H     Origin -2 of word address list
4E3C 09          ADD     HL,BC        Add index * 2 for required phrase
4E3D 5E          LD      E,(HL)       E = LSB of addr for required phrase
4E3E 23          INC     HL           Bump to address of MSB
4E3F 56          LD      D,(HL)       D = MSB of addr for required phrase
4E40 23          INC     HL           Bump to LSB of addr for next phrase
4E41 7E          LD      A,(HL)       Fetch it
4E42 23          INC     HL           Bump to MSB of addr for next phrase
4E43 66          LD      H,(HL)       Fetch it
4E44 6F          LD      L,A          HL = text addr for following phrase
4E45 B7          OR      A            Clear carry
4E46 ED52        SBC     HL,DE        Addr of next phrase (one we want)

4E48 45          LD      B,L          Gives # of chars in phrase we need
4E49 EB          EX      DE,HL        HL = text addr for required phrase
4E4A 7E          LD      A,(HL)       Fetch a character from phrase
4E4B 23          INC     HL           Bump to next char in phrase
4E4C CD3300      CALL    0033H        Display phrase character
```

Listing Continued . . .

. . . Continued Listing

```
4E4F 10F9       DJNZ    4E4AH       Loop till all of phrase displayed
4E51 3E20       LD      A,20H       Follow phrase
4E53 CD3300     CALL    0033H       with a space
4E56 E1         POP     HL          Restore phrase index address
4E57 7E         LD      A,(HL)      Refetch index
4E58 23         INC     HL          Bump to the next index in case
                                    this is not the end
4E59 07         RLCA                Terminating bit to CARRY flag
4E5A 30D5       JR      NC,4E31H    Jump if not end of phrase
                                    list. Process next phrase
```

Summarized, the code from **4E2A - 4E2F** forms the address of the error message list in the HL register using the error code from the parameter byte as an index into the list beginning at **4F84**. The code from **4E31 - 4E3C** forms the address of the word address list in the HL registers, and the code from **4E3D - 4E46** computes the beginning address of the error message and its length. The loop from **4E4A - 4E4F** displays a single word of the error message. The code from **4E57 - 4E5A** tests the current word index to determine if the last word of the message has been dislayed. If not, control goes to **4E31**, where the address of the next word is computed before it is displayed.

When the end of the message is reached (a word index with bit 8 on is detected), control goes to **4E5C**.

After the error message has been displayed, bit 6 of the parameter byte is tested and SYS4 exits, or the next message is displayed. The following code is used:

Figure 7.8 *SYS4 Code From 4E5C to 4E70*

```
4E5C F1         POP     AF          Restore error code
4E5D F5         PUSH    AF          Save error code
4E5E CB77       BIT     06H,A       Test if file error code
4E60 CC734E     CALL    Z,4E73H     Display file name if file error
4E63 3E0D       LD      A,0DH       Else scroll one line
4E65 CD3300     CALL    0033H       by displaying a carriage return
4E68 F1         POP     AF          Restore error code in A,
4E69 C1         POP     BC          then restore BC
4E6A D1         POP     DE          DE
4E6B E1         POP     HL          and HL
4E6C B7         OR      A           Set status flag for return
                                    Test for System or caller return
4E6D FA0000     JP      M,0000H     Return to caller if requested
4E70 C33040     JP      4030H       Else return to system
```

Control passes to **4E73** if Bit 6 of the parameter byte is zero. In this case, an error with a file is assumed, and the message '<FILE=, and REFERENCED' will probably be displayed. The exception condition that may cause the message 'DEVICE= ..... ' to appear will be detected in the next section.

The code from **4E73** through **4F33** (the end code in SYS4) can be divided into two sections. The first section deals with the exception cases described earlier, while the second section produces the file messages. Some of the code is shared between the sections.

Beginning at **4E73** and continuing through **4EAD** is the controlling code which determines if the exception conditions are present, and if so, generates (but does not display) the '<DEV' message. The code from **4E73 - 4E84** is common to both sectons:

Figure 7.9 *SYS4 Code From 4E73 to 4EAD*

```
4E73 E5          PUSH    HL              Save last phrase index address
4E74 21474F      LD      HL,4F47H        Address of ***
4E77 CD6744      CALL    4467H           Display closing ***
4E7A E1          POP     HL              Restore last phrase index addr
4E7B DDE5        PUSH    IX              Save current system DCB address
4E7D DD2A0A43    LD      IX,(430AH)      Load DCB addr for file with error
4E81 2B          DEC     HL              Position to terminating character
                                         in phrase index list
4E82 CB76        BIT     06H,(HL)        Test if file name in DCB
4E84 201A        JR      NZ,4EA0H        Jump if file name in DCB
4E86 DD4E06      LD      C,(IX+06H)      C = drive number
4E89 DD4607      LD      B,(IX+07H)      B = directory sector number
4E8C DDCB007E    BIT     07H,(IX+00H)    Test if file OPEN
4E90 2032        JR      NZ,4EC4H        If so, go read dir - copy name/ext
4E92 DDE1        POP     IX              Else, clear stack
4E94 ED43554F    LD      (4F55H),BC      Save device code in DEVICE = xx msg
4E98 214B4F      LD      HL,4F4BH        Address of DEVICE = xx msg
4E9B CD6744      CALL    4467H           Display message
4E9E 1866        JR      4F06H           Go display REFERENCED AT message
                                         and return to caller
4EA0 DD7E00      LD      A,(IX+00H)      Test for special DCB code
4EA3 FE2A        CP      2AH             First character must be *
4EA5 2008        JR      NZ,4EAFH        Jump if not. Name is in DCB
4EA7 DD4E01      LD      C,(IX+01H)      Special DCB
4EAA DD4602      LD      B,(IX+02H)      Bytes 2 - 3 contain device code
4EAD 18E3        JR      4E92H           Go print DEVICE=XX msg
```

Notice that the DCB address of the file in error is loaded from **430A**. A service routine in the nucleus saves the DCB and sector buffer address at **430A** and **430C** when processing user level I/0 calls. System level calls, such as those used for overlay loading and accessing the GAT and HIT sectors, do not cause those locations to be updated.

Code for the second section runs from **4EAF** through **4F33**. Portions of this code are used by the first section to display the '<DEV ..... ' message.

The code in this section determines the file name and sector buffer address and incorporates them into the remainder of the error message. Two possibilities exist for locating the file name. If the file is unopened, the name is assumed to be in the DCB; the code from **4EAF - 4EC2** deals with this case.

If the file has been opened, the name must be fetched from the directory. The code from **4EC4 - 4EF8** performs that operation.

Finally, the code from **4EF9 - 4F33** displays the remainder of the error messages and returns to **4E63** (remember, this stretch of code was entered by a CALL) where control is returned to the caller.

Figure 7.10 *SYS4 Code From 4EAF to 4F33*

```
4EAF DDE5       PUSH    IX          Save DCB address belonging to
                                    file with error
4EB1 E1         POP     HL          HL = DCB address with file name
4EB2 11604F     LD      DE,4F60H    Address of file name in
                                    <file = NNN...> msg
4EB5 011800     LD      BC,0018H    Max number of chars in name/ext:D
4EB8 7E         LD      A,(HL)      Fetch a char from file name
4EB9 FE03       CP      03H         Test for end of list
4EBB 283C       JR      Z,4EF9H     Jump if end of list
4EBD 23         INC     HL          Else bump to next char in list
4EBE 12         LD      (DE),A      Move char to <file = NNN...> msg
4EBF 13         INC     DE          Bump to next pos in file message
4EC0 10F6       DJNZ    4EB8H       Loop till file/ext:drive copied
                                    into <file = ...> message
4EC2 1835       JR      4EF9H       Go display <file = ...> message
4EC4 CDC14A     CALL    4AC1H       Read dir entry for file with error
4EC7 010500     LD      BC,0005H    Add offset to get address
4ECA 09         ADD     HL,BC       of file name in directory
4ECB 11604F     LD      DE,4F60H    Addr of name in <file = ...> msg
4ECE 0608       LD      B,08H       Max. no. chars to copy from dir
4ED0 7E         LD      A,(HL)      Get file name char from directory
4ED1 FE20       CP      20H         Is it blank (end of name reached)
4ED3 2805       JR      Z,4EDAH     Jump if end of name
4ED5 2C         INC     L           Bump to next char if not
4ED6 12         LD      (DE),A      Save this char in <file = ...> msg
4ED7 13         INC     DE          Bump to next pos in <file = ...> msg
4ED8 10F6       DJNZ    4ED0H       Loop till 8 characters copied
                                    or end of name found
```

Listing Continued . . .

. . . Continued Listing

```
4EDA 3E2F       LD      A,2FH       Then load an ASCII "/"
4EDC 12         LD      (DE),A      Put after name in <file = ...> msg
4EDD 13         INC     DE          Pos to addr of ext in <,file> msg
4EDE 48         LD      C,B         C = number of characters
                                    remaining in name field
4EDF 0600       LD      B,00H       So BC can be added to HL
4EE1 09         ADD     HL,BC       compute beginning addr of ext
4EE2 0603       LD      B,03H       Max no. chars in extension
*
*
*
4EE4 7E         LD      A,(HL)      .   Fetch a char from ext
4EE5 2C         INC     L           .   Bump to next char in ext
4EE6 FE20       CP      20H         .   If char blank, we're done
4EE8 2804       JR      Z,4EEEH     .   Jump if end of ext found
4EEA 12         LD      (DE),A      .   Move ext char to message
4EEB 13         INC     DE          .   Bump message store address
4EEC 10F6       DJNZ    4EE4H       .   Loop till extension copied
4EEE EB         EX      DE,HL       HL = addr of drive # in <file = ...>
4EEF 363A       LD      (HL),3AH    terminate filename/ext
4EF1 23         INC     HL          with a :
4EF2 DD7E06     LD      A,(IX+06H)  Then fetch drive from DCB
4EF5 C630       ADD     A,30H       and convert it to ASCII
4EF7 77         LD      (HL),A      Save in <file = ...> message
4EF8 23         INC     HL          Bump to pos after drive # in msg
4EF9 363E       LD      (HL),3EH    Save closing >
4EFB 23         INC     HL          Bump past > in msg
4EFC 360D       LD      (HL),0DH    Terminate with carriage return
4EFE DDE1       POP     IX          Restore current DCB address
4F00 21594F     LD      HL,4F59H    Address of <file = XXX/XX:d> msg
4F03 CD6744     CALL    4467H       Display message
4F06 21704F     LD      HL,4F70H    Address of REFERENCED AT message
4F09 CD6744     CALL    4467H       Display message
4F0C 2A0C43     LD      HL,(430CH)  HL = sector buffer address
                                    associated with file
4F0F 2B         DEC     HL          Decrement buffer
4F10 2B         DEC     HL          Address
4F11 2B         DEC     HL          By three
4F12 CD1B4F     CALL    4F1BH       Convert addr to ASCII and display
4F15 21824F     LD      HL,4F82H    Address of ' string (terminate
                                    buffer address display)
4F18 C36744     JP      4467H       Display ending ', ret to caller
4F1B 7C         LD      A,H         Convert MSB of buffer addr first
4F1C CD204F     CALL    4F20H       Call routine to convert and
                                    display value in A reg
4F1F 7D         LD      A,L         Now convert/display LSB of buff addr
4F20 F5         PUSH    AF          Save value to be converted
4F21 1F         RRA                 Right shift hex
4F22 1F         RRA                 value four times
4F23 1F         RRA                 This is the same
4F24 1F         RRA                 as dividing it by 16
4F25 CD294F     CALL    4F29H       Then convert binary hex value
                                    to ASCII and display
4F28 F1         POP     AF          Restore original value
```

Listing Continued . . .

. . . Continued Listing

```
4F29 E60F       AND     0FH             Isolate binary field to convert
4F2B C630       ADD     A,30H           Gives ASCII equivalent of binary
4F2D FE3A       CP      3AH             Test if value exceeds 9
4F2F 3802       JR      C,4F33H         Jump if not, value less than 10
4F31 C607       ADD     A,07H           Add 7 to get A-F ASCII value
4F33 C33300     JP      0033H           Display char. Return to caller
```

8

8.0 SYS5/SYS (DEBUG)

SYS5/SYS is the TRSDOS DEBUG utility program. This utility may be used for debugging assembly language programs. Since it is an overlay to the nucleus module, the user need not be concerned with its load address. However, this also means that it cannot be used for debugging system overlays because it occupies the same address space.

The DEBUG command is processed by SYS1. When a DEBUG,ON command is recognized, a DEBUG flag in the system condition byte (see Section 5.1) is set. A DEBUG,OFF command causes the DEBUG flag to be cleared.

DEBUG is executed after a DEBUG,ON command has been processed, and when one of the following conditions is true:

1. An active BREAK key is detected (during system interrupt processing).

2. At the completion of loading a utility, or user assembly language program.

3. When a TRAP instruction is executed. Because the Z-80 has no TRAP instruction an RST 30 is used to effect a TRAP.

Debug occupies locations **4E00 - 51FF**. In addition to its own internal data structures and lists, DEBUG uses locations **4065 - 407C**, which are reserved for this purpose, to save the program context (all of the program registers at the time of the trap) plus the trap addresses and their original contents. The program context could be saved in a local storage area; however, that would increase the size of DEBUG and push it over its current boundary of **51FF**.

The trap addresses and their original contents must be saved in the low memory, or some such area that would not be destroyed when DEBUG is reloaded, since it is possible for DEBUG to be overwritten by other system overlays while TRAP'ing through a program. This allows DEBUG to restore instructions that were replaced by TRAP instructions previously.

Local data structures in DEBUG consist of a text string used in the X display modes and an instruction interpretation list. This list is used to determine instruction length and conditional branch conditions during the single cycle execution mode.

The structure of DEBUG is quite straightforward. Subroutines corresponding to each of the options are entered by direct jumps. A common return address is pushed onto the stack before jumping to any of the option subroutines. There are several support subroutines which are shared by the options subroutines. Among them are:

1. A subroutine which stores a trap instruction at a specified address. The address, and its original contents are saved in the trap list in low memory (**4FCA - 4FDA**).

2. A subroutine which displays the contents of 16 bytes beginning at the address specified in the HL register (**5131 - 5189**).

3. A subroutine which reads one character from the keyboard and returns the character, as well as a status indicating the type of character, e.g., hex. digit, space, or carriage return (**518A - 51A2**).

4. A subroutine which interprets a hex value (**51A3 - 51CF**).

5. A subroutine which converts a hex value. Note, this subroutine has multiple entry points (**51D0 - 51FE**).

8.1 DEBUG Entry Processing

Upon entry, DEBUG preserves the context at the time the TRAP instruction (RST 30) was executed, restores the original instructions to all locations containing TRAP instructions and enters the X display mode.

The context is saved in low memory at addresses **4065 - 407C**. The format of the context saved at those addresses is shown below:

Figure 8.1 *DEBUG Context Save Area*

ADDRESS	CONTENTS
4065-66	AF
4067-68	BC
4069-6A	DE
406B-6C	HL
406D-6E	AF'
406F-70	BC'
4071-72	DE'
4073-74	HL'
4075-76	IX
4077-78	IY
4079-7A	SP
407B-7C	PC

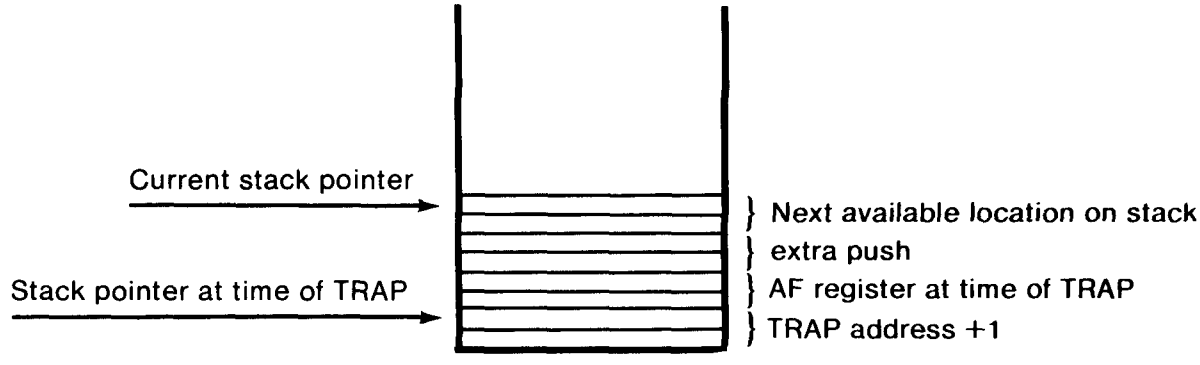
In addition to the context save area, low memory holds the trap address list and current display address at the locations shown below.

Figure 8.2 *Trap Address List*

Address	Contents
405D - 5E	1st trap address
405F	Original contents of trap address
4060 - 61	2nd trap address
4062	Original contents of trap address
4063 - 64	Display address

When DEBUG is entered, the stack has the following configuration:

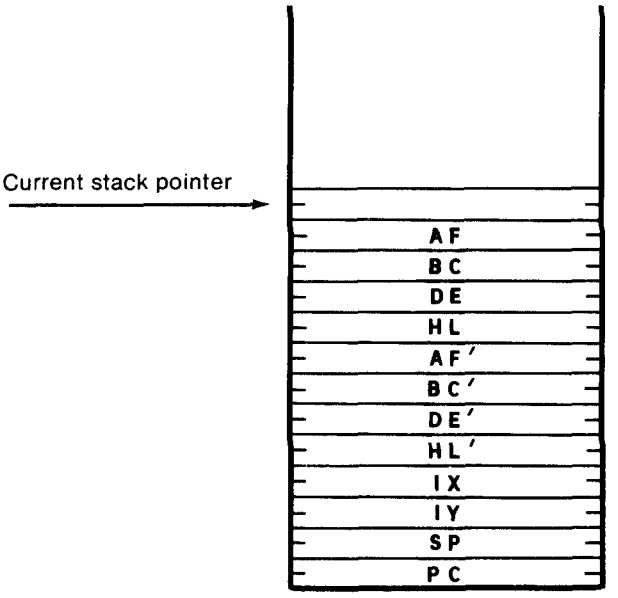
Figure 8.3 *Stack Configuration When DEBUG Entered*



Stack Configuration When DEBUG Entered

The context at the time of the TRAP, except for the AF and PC registers, is still in the registers. This context is saved on the stack, giving it the following configuration:

Figure 8.4 *Stack Configuration After DEBUG Entry Processing*



STACK CONFIGURATION AFTER DEBUG ENTRY PROCESSING

Note, the stack pointer at the time of the TRAP replaced the value for the AF register, which has been moved to the end of the stack frame. The following instructions are used to save the register context and copy it to the communication area:

Figure 8.5 *SYS5 Code From 4E00 to 4E22*

```
4E00 F1          POP     AF          Clear unnec. PUSH, restore AF
4E01 F1          POP     AF          Then create a 'hole' for SP
4E02 F5          PUSH    AF          Will be filled in at 4E1F
4E03 FDE5        PUSH    IY          Save IX, no alternate
4E05 DDE5        PUSH    IX          Save IY, no alternate
*
*                Save Context of Alternate Registers
*
4E07 08          EX      AF,AF'      Switch to alternate
4E08 D9          EXX                 Register set
4E09 E5          PUSH    HL          Save alternate HL
4E0A D5          PUSH    DE          Save alternatete DE
4E0B C5          PUSH    BC          Save alternate BC
4E0C F5          PUSH    AF          Save alternate AF
```

Listing Continued . . .

. . . Continued Listing

```
4E0D 08        EX      AF,AF'      Switch to primairy
4E0E D9        EXX                 Register set
4E0F E5        PUSH    HL          Save main HL
4E10 D5        PUSH    DE          Save main DE
4E11 C5        PUSH    BC          Save main BC
4E12 F5        PUSH    AF          Save main AF
4E13 210000    LD      HL,0000H    Clear HL so we can load SP into HL
4E16 39        ADD     HL,SP       HL points to start of saved context
4E17 116540    LD      DE,4065H    DEBUG context save area addr.
4E1A 011800    LD      BC,0018H    Number of bytes in saved context
4E1D EDB0      LDIR                Copy from stack to low mem. save area
4E1F 227940    LD      (4079H),HL  Save value of SP at time of trap
4E22 F9        LD      SP,HL       Restore SP value at time of trap
```

Figure 8.6 *SYS5 Code From 4E23 to 4E49*

```
4E23 2A7B40    LD      HL,(407BH)  Addr of PC register +1 at trap time.
4E26 2B        DEC     HL          Back-up to addr. of trap instr.
                                   to get continue addr.
4E27 7E        LD      A,(HL)      Fetch instr. that caused trap
4E28 FEF7      CP      0F7H        Was it a RST 30 (trap instr.)
4E2A 2003      JR      NZ,4E2FH    Jump if not, may have been entered
                                   because of BREAK key hit
4E2C 227B40    LD      (407BH),HL  Yes, update PC reg. in context area
4E2F 215D40    LD      HL,405DH    Load trap address table
*
*       Trap address table contains original contents of
*       address where traps have been stored. Format of
*       entries is as follows:
*
*                         XX    LSB of trap addr
*                         XX    MSB of trap addr
*                         XX    Original value
*
*     Restore Original Contents of Locations That Have Traps
*
4E32 0602      LD      B,02H       .  Number of entries in table.
4E34 AF        XOR     A           .  Zero A
4E35 0E02      LD      C,02H       .  Unnecessary instruction
4E37 5E        LD      E,(HL)      .  Get LSB of possible trap addr (TA)
4E38 77        LD      (HL),A      .  Zero LSB of TA from list
4E39 23        INC     HL          .  Bump to MSB of TA
4E3A 56        LD      D,(HL)      .  Fetch MSB of TA from list
4E3B 77        LD      (HL),A      .  Zero MSB of TA
4E3C 23        INC     HL          .  Bump to original value of TA
4E3D 7B        LD      A,E         .  Combine LSB and MSB
4E3E B2        OR      D           .  Of trap address
4E3F 2807      JR      Z,4E48H     .  Jump if no entry, else check
                                   .  for valid entry
```

Listing Continued . . .

. . . Continued Listing

```
4E41 1A        LD      A,(DE)      .  Fetch contents of trap addr.
4E42 FEF7      CP      0F7H        .  Is it a RST 30 (trap instruction)
4E44 2002      JR      NZ,4E48H    .  No, check next entry
4E46 7E        LD      A,(HL)      .  Yes, restore original instruction
                                   .  Fetch saved instruction
4E47 12        LD      (DE),A      .  and restore to location
                                   .  that contains trap instruction
4E48 23        INC     HL          .  Bump to next entry in list
4E49 10E9      DJNZ    4E34H       .  Loop till all instr. restored
```

Next, the main loop in DEBUG is entered. This loop drives the display and waits for the next command character.

When an input character is received, it is compared, and if there is a match, control branches to a subroutine associated with the command character. After the command has been processed, control returns to **4E4B**, which is the beginning of the main loop. Following is the code used in the main loop:

Figure 8.7 *SYS5 Code From 4E4B to 4E98*

```
*
*                  Input Command Loop
*
4E4B ED7B7940  LD      SP,(4079H)  Restore context table value to SP
4E4F CDCF4E    CALL    4ECFH       Display register values (X display)
4E52 21C03F    LD      HL,3FC0H    Addr. of next to last row of display
4E55 222040    LD      (4020H),HL  Save as input line display area
4E58 CD8A51    CALL    518AH       Get a char. of input
4E5B FE47      CP      47H         Test for G
4E5D CA804F    JP      Z,4F80H     Jump if G command (Go command)
4E60 214B4E    LD      HL,4E4BH    Return addr. for all other commands
4E63 E5        PUSH    HL          To the stack
4E64 FE53      CP      53H         Test for S
4E66 2832      JR      Z,4E9AH     Jump if S command
4E68 FE3B      CP      3BH         Not S, test for;
4E6A 2842      JR      Z,4EAEH     Jump if semi-colon (increment
                                   memory display 1 page)
4E6C FE2D      CP      2DH         Not ";" test for "-"
4E6E 2856      JR      Z,4EC6H     Jump if dash (decrement memory
                                   display 1 page)
4E70 FE41      CP      41H         Not dash, test for A
4E72 2857      JR      Z,4ECBH     Jump if A command (ASCII display)
4E74 FE43      CP      43H         Not A, test for C
4E76 280A      JR      Z,4E82H     Go if C cmd (Single step with calls)
4E78 FE44      CP      44H         Not C, test for D
4E7A 282C      JR      Z,4EA8H     Jump if D command (Dislay memory)
4E7C FE48      CP      48H         Not D, test for H
4E7E 284B      JR      Z,4ECBH     Jump if H cmd (Hexadecimal display)
4E80 FE49      CP      49H         Not H, test for I and go if
4E82 CA5D50    JP      Z,505DH     I command (Single step next instr)
```

Listing Continued . . .

. . . Continued Listing

```
4E85 FE4D      CP      4DH         Not I, test for M (Enter
                                   memory modification mode)
4E87 CADB4F    JP      Z,4FDBH     Jump if M command
4E8A FE52      CP      52H         Not M, test for R (mod. reg. contents)
4E8C CA1150    JP      Z,5011H     Jump if R command
4E8F FE55      CP      55H         Not R, test for U (Dynamic display)
4E91 280B      JR      Z,4E9EH     Jump if U command (Dynamic display)
4E93 FE58      CP      58H         Not U, test for X (Return
                                   to primary display)
4E95 2802      JR      Z,4E99H     Jump if X command to
4E97 EF        RST     28H         Process input char as overlay request
4E98 C9        RET                 Then return to 4E4B
```

8.2 Display Formats

There are two display formats available to DEBUG users. One, called the X, or register, display is shown below. The other, called the memory display, is also shown below.

Figure 8.8a *Register Display*

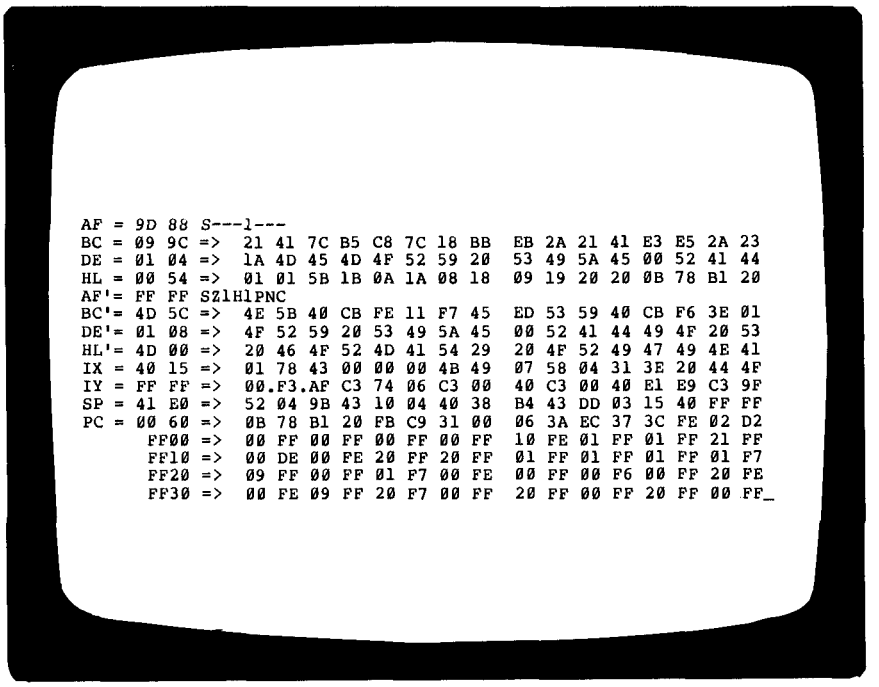
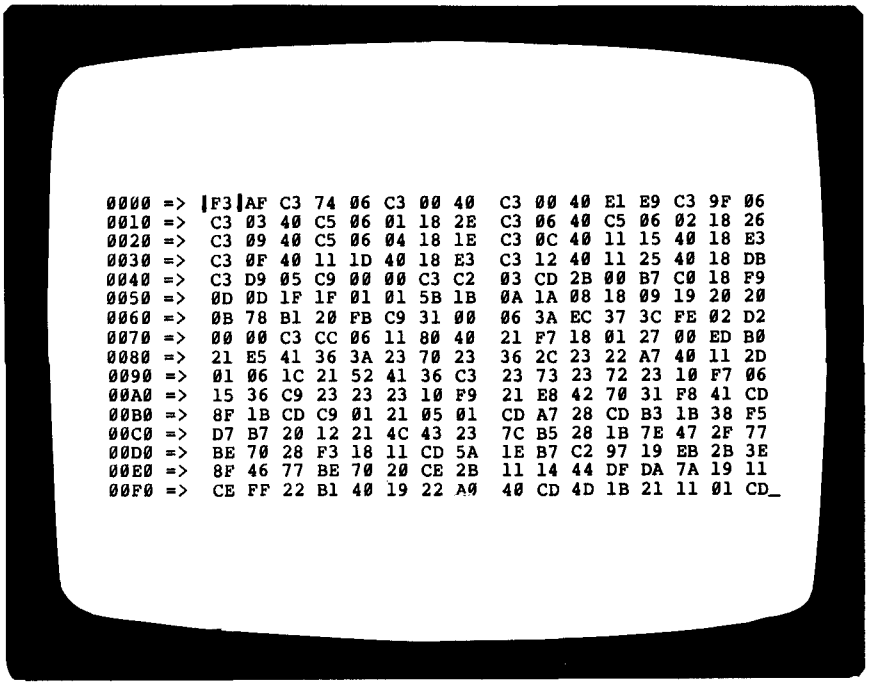


Figure 8.8b *Register Display*



Either display may be in ASCII or hexadecimal mode. The mode and format are selected by the commands:

A - ASCII mode
H - HEXADECIMAL mode
X - Register display format
S - Memory display format

The beginning memory address for both display formats is specified using the DXXXX command, where XXXX is the starting address for the display. When in the register dislay format, the memory display begins at the exact address specified in the D command. When in the memory display format, the address is rounded up to the nearest 256-byte page boundary. Display addresses may be scrolled forward or backward 256 bytes through use of the ';' '+' and '-' commands.

The display subroutine begins at **4ECF** and continues through **4F52**. A text list used for the register display begins at **4F54** and ends at **4F7F**. Support subroutines used by the main display routine begin at **5131** and **51D0**. The use of these subroutines by the main display loop is shown below.

Figure 8.9 *SYS5 Code From 4ECF to 4F7F*

```
*
*                    Register display
*
4ECF 3A5E40     LD      A,(405EH)   Get display mode flag
4ED2 B7         OR      A           Set status flags for display size
4ED3 2070       JR      NZ,4F45H    Jump if not register display
4ED5 3E1C       LD      A,1CH       Home cursor command
4ED7 CD3300     CALL    0033H       Give video driver home cursor command
4EDA 216540     LD      HL,4065H    Address of saved register context
4EDD E5         PUSH    HL          Save context address
4EDE 21544F     LD      HL,4F54H    Address of register name text
4EE1 060C       LD      B,0CH       No. of register values to display
4EE3 1805       JR      4EEAH       Go display register context
*
*       Display register values plus the contents of the
*                addresses contained therein.
*
4EE5 3E0D       LD      A,0DH       .  Use a carriage return to
4EE7 CD3300     CALL    0033H       .  Skip to next line of display
4EEA CDF651     CALL    51F6H       .  Display reg. name pointed to by HL
4EED E3         EX      (SP),HL     .  Addr. of next reg. name
                                    .  to stack. Context addr. for
4EEE 5E         LD      E,(HL)      .  Register to HL. Load LSB of
                                    .  reg. value from context list
4EEF 23         INC     HL          .  Bump to MSB of context list
4EF0 56         LD      D,(HL)      .  Fetch MSB of reg. value
4EF1 23         INC     HL          .  Bump to addr. of next reg. set
4EF2 E5         PUSH    HL          .  Save addr. in context list
4EF3 EB         EX      DE,HL       .  Value for that reg. to HL
4EF4 3E3D       LD      A,3DH       .  ASCII value for =
4EF6 CD3300     CALL    0033H       .  Display =
4EF9 CDF251     CALL    51F2H       .  Display a blank following =
4EFC 7C         LD      A,H         .  A = MSB of context value
4EFD CDEF51     CALL    51EFH       .  Convert to ASCII hex and display
4F00 7D         LD      A,L         .  A = LSB of context value
4F01 CDEF51     CALL    51EFH       .  Convert to ASCII hex and display
4F04 78         LD      A,B         .  A = remaining number of register
                                    .  values to be displayed
4F05 E60B       AND     0BH         .  Test if index is exactly 8
4F07 FE08       CP      08H         .  and if not, display status flags
4F09 201C       JR      NZ,4F27H    .  Jump if not displaying AF reg.
4F0B 4D         LD      C,L         .  Status flags to C register
4F0C C5         PUSH    BC          .  Save no. of reg. to be displayed
4F0D 21784F     LD      HL,4F78H    .  Addr of ASCII chars for status bits
4F10 0608       LD      B,08H       .  B = no. of stat bits to examine
4F12 CB21       SLA     C           .  Shift a status bit into CARRY
4F14 7E         LD      A,(HL)      .  Get letter for status condition
4F15 3802       JR      C,4F19H     .  Jump if status bit is on
                                    .  and display letter, else
4F17 3E2D       LD      A,2DH       .  ASCII value for -
```

Listing Continued . . .

. . . Continued Listing

```
4F19 CD3300     CALL    0033H       .   Display a - (or letter if
                                    .   condition is true)
4F1C 23         INC     HL          .   Bump to next letter in status list
4F1D 10F3       DJNZ    4F12H       .   Loop till all status bits
                                    .   tested and displayed
4F1F C1         POP     BC          .   restore no. reg. left to display
4F20 3EEC       LD      A,0ECH      .   Comparison code for 44 blanks
4F22 CD3300     CALL    0033H       .   Display 44 blanks
4F25 1803       JR      4F2AH       .   Skip over => value display
4F27 CD3151     CALL    5131H       .   Display => followed by
                                    .   contents of addr. in reg.
4F2A E1         POP     HL          .   Restore addr next register set
4F2B E3         EX      (SP),HL     .   Put address on stack
4F2C 10B7       DJNZ    4EE5H       .   Loop till all registers displayed
4F2E E1         POP     HL          Clear stack
4F2F 2A6340     LD      HL,(4063H)  Load beginning display address
```

The subroutine at **5131** is called from the register display routine to display the value 1, value 2, ... value 16 portion of the display. The code for this subroutine is shown below.

Figure 8.10 *SYS5 Code From 5131 to 5188*

```
5131 C5         PUSH    BC          Preserve caller's BC
5132 3E3D       LD      A,3DH       ASCII value for =
5134 CD3300     CALL    0033H       Display =
5137 3E3E       LD      A,3EH       ASCII value for >
5139 CD3300     CALL    0033H       Display >
513C CDF251     CALL    51F2H       Display a space
513F 0610       LD      B,10H       Number of values to display.
5141 CD6551     CALL    5165H       If addr to modify is being
                                    displayed bracket it with bars
5144 3A5D40     LD      A,(405DH)   Fetch display mode flag
5147 FE41       CP      41H         Test for A
5149 2013       JR      NZ,515EH    Jump if not alphabetical
514B CDF251     CALL    51F2H       Display a blank
514E 7E         LD      A,(HL)      Fetch a character
514F FE20       CP      20H         Test for control code
                                    (compare to space)
5151 3804       JR      C,5157H     Jump if control code. Display a .
5153 FEC0       CP      0C0H        Test for compression code
5155 3802       JR      C,5159H     Jump if not a compression code
5157 3E2E       LD      A,2EH       ASCII value for .
5159 CD3300     CALL    0033H       Display character or .
515C 23         INC     HL          Bump to next character in list
```

Listing Continued . . .

. . . Continued Listing

```
515D AF          XOR     A            Clear status to skip CALL below
515E C4D051      CALL    NZ,51D0H     Display a hex value
5161 10DE        DJNZ    5141H        Loop till 16 values displayed
5163 C1          POP     BC           Restore caller's BC
5164 C9          RET                  Return to caller
*
*       Put Display Bars Around Location Being Modified
*
5165 ED5B6040    LD      DE,(4060H)   Load 2nd trap address into DE
5169 13          INC     DE           Bump to addr of following instruction
516A E5          PUSH    HL           Save current display address
516B AF          XOR     A            Clear CARRY for SBC operation
516C ED52        SBC     HL,DE        Compare display and modify address
516E 3E95        LD      A,95H        ASCII code for "left bar"
5170 280E        JR      Z,5180H      If display = modify addr.
                                      disp. left bar
5172 CD8451      CALL    5184H        If 8 values disp. add an extra space
5175 23          INC     HL           Bump display address
5176 7D          LD      A,L          Then combine
5177 B4          OR      H            LSB and MSB
5178 E1          POP     HL           Restore current display address
5179 3EAA        LD      A,0AAH       ASCII code for right bar
517B CA3300      JP      Z,0033H      If disp. addr = modify address
                                      Display closing right bar
517E 1808        JR      5188H        Display space between values and ret
5180 CD3300      CALL    0033H        Display left bar
5183 E1          POP     HL           Restore display address
5184 78          LD      A,B          Get no. of lines still to display
5185 FE08        CP      08H          If half a line (8) left, add a space
5187 C0          RET     NZ           Exit if not half way point yet
5188 1868        JR      51F2H        Go display a blank and ret to caller
```

A second set of display subroutines is shown below. These subroutines have multiple entry points which are identified as ENTRY 1, ENTRY 2, . Each entry point is specific to a particular format and mode.

Figure 8.11 *SYS5 Code From 51D0 to 51FE*

```
51D0 7E Entry 1  LD      A,(HL)       Fetch an 8-bit hex value
51D1 23          INC     HL           Bump to address of next 8-bit value
51D2 1805        JR      51D9H        Go convert hex to ASCII and ret
51D4 7C Entry 2  LD      A,H          MSB of value to convert and display
51D5 CDD951      CALL    51D9H        Display 2 hex digits (MSB), then
51D8 7D          LD      A,L          get LSB and display other 2 hex digits
51D9 F5          PUSH    AF           Save value to be displayed
51DA 1F          RRA                  Right shift value
51DB 1F          RRA                  four
51DC 1F          RRA                  places
51DD 1F          RRA                  to isolate and convert upper
51DE CDE251      CALL    51E2H        4 bits to hex ASCII and display
51E1 F1          POP     AF           Restore value being displayed
```

Listing Continued . . .

. . . Continued Listing

```
51E2 E60F       AND     0FH         Isolate lower 4 bits of value
51E4 C630       ADD     A,30H       Gives ASCII value for digit
51E6 FE3A       CP      3AH         Test for hex value A-F
51E8 3802       JR      C,51ECH     Jump if digit 0-9 else
51EA C607       ADD     A,07H       get ASCII value for A-F
51EC C33300     JP      0033H       .   Display char in A and return
51EF CDD951     CALL    51D9H       .   Display 2 hex digits
51F2 3E20       LD      A,20H       .   ASCII code for a blank
51F4 18F6       JR      51ECH       .   Display blank and return
51F6 CDFC51     CALL    51FCH       .   Get char from list in HL
                                    .   Display char and ret to 51F9
51F9 CDFC51     CALL    51FCH       .   Get next char from list
                                    .   Display and Ret to 51FC
51FC 7E         LD      A,(HL)      .   Get char from list addr in HL
51FD 23         INC     HL          .   Bump to next char. in list
51FE 18EC       JR      51ECH       .   Display char, ret.
                                    .   to caller of 51FC
```

8.3 'A,' 'H,' 'X,' 'S,' ';' and '-' Commands

These commands select a display format or mode. The code for the display driver has been discussed previously. This section will describe the code executed for the 'A,' 'H,' 'X,' ';' and '-' commands.

Code used for the 'A' and 'H' commands follows:

Figure 8.12 *SYS5 Code From 4ECB to 4ECE*

```
*
*                   A/H Command.
*
4ECB 325D40     LD      (405DH),A   Save display mode value
4ECE C9         RET                 Return to main display loop
```

Code for the 'X' and 'S' commands follows:

Figure 8.13 *SYS5 Code From 4E99 to 4E9D*

```
*
*           X Command. Return to Primary Display.
*
4E99 AF         XOR     A           Clear A-reg.
4E99 325E40     LD      (405EH),A   Signal short main display)
4E9D C9         RET                 Return to 4E4B
```

Code for the ';' and '-' command follows:

Figure 8.14 *SYS5 Code From 4EAE to 4EC9*

```
*
*   ; or - Command. Adjust Display Address (Forward or Backward)
*
4EAE 014000     LD      BC,0040H    Amt. to advance display (1 page);
4EB1 2A6340     LD      HL,(4063H)  Current beginning display address
4EB4 3A5E40     LD      A,(405EH)   Get display mode flag
4EB7 B7         OR      A           Set status for display mode
4EB8 2807       JR      Z,4EC1H     Jump if short display
4EBA 0E00       LD      C,00H       Clear LSB of adjustment address
4EBC 78         LD      A,B         Fetch MSB of adjustment addr and
4EBD B7         OR      A           Set flags for MSB of adjustment addr
4EBE 2001       JR      NZ,4EC1H    Jump if negative adjustment
4EC0 04         INC     B           Gives 100 (positive adjustment)
4EC1 09         ADD     HL,BC       Compute new display address, then
4EC2 226340     LD      (4063H),HL  save beginning display address
4EC5 C9         RET                 Ret to 4E4B
*
*               - Command. Decrement Memory Display
*
4EC6 01C0FF     LD      BC,0FFC0H   Adj. value for previous page
4EC9 18E6       JR      4EB1H       Go update display address
```

8.4 'D' Command

The 'D' command initializes the beginning display address, which is kept at **4063 - 4064**, to the hexidecimal value following the 'D' character. Following is the code used for processing this command:

Figure 8.15 *SYS5 Code From 4EA8 to 4EAC*

```
*
*                 D Command. Dislay Memory.
*
4EA8 CDA351     CALL    51A3H       Get beginning addr. from keyboard
4EAB C8         RET     Z           Rtn. to 4E4B if any error
4EAC 1814       JR      4EC2H       Else save starting display address
```

Support subroutines required for this command are shown below:

Figure 8.16 *SYS5 Code From 51A3 to 51CF*

```
*
*                       Input a hex value
*
51A3 CD8A51     CALL    518AH       Wait for next character
51A6 C8         RET     Z           Ret to caller if it's blank or comma
51A7 210000     LD      HL,0000H    Zero accum. to contain hex val on exit
51AA CDBF51     CALL    51BFH       Get binary value if it was a hex digit
51AD DA4B4E     JP      C,4E4BH     Jump if not a hex digit (0-9,A-F)
51B0 29         ADD     HL,HL       Digit times 2
51B1 29         ADD     HL,HL       Digit times 4
51B2 29         ADD     HL,HL       Digit times 8
51B3 29         ADD     HL,HL       Digit times 16. In case more
51B4 B5         OR      L           than 2 digits are entered
51B5 6F         LD      L,A         Only last 2 will be used
51B6 CD8A51     CALL    518AH       Get next digit and if not a blank
51B9 20EF       JR      NZ,51AAH    go test char, accumulate hex value
51BB 1F         RRA                 CARRY flag to Bit 7
51BC CE81       ADC     A,81H       Will set CARRY if carriage return
51BE C9         RET                 Return to caller
*
*      Test character. Return with CARRY if not 0 - 9, or A - F
*                      (hex digit)
*
51BF D630       SUB     30H         Compare with ASCII zero
51C1 D8         RET     C           Ret if less than 0. Obviously not hex
51C2 C6E9       ADD     A,0E9H      Test for a letter > F
51C4 D8         RET     C           Return if > F. Not a hex digit
51C5 C606       ADD     A,06H       Test for letter (A-Z)
51C7 3803       JR      C,51CCH     Jump if letter
51C9 C607       ADD     A,07H       Not letter, digit 0-9
51CB D8         RET     C           Unnecessary instruction
51CC C60A       ADD     A,0AH       Compute final binary value
51CE B7         OR      A           Set status flags for value
51CF C9         RET                 Return to caller
```

8.5 'M' Command

This command modifies a memory location. It is more complex than one would think it ought to be because it must update any display line which contains the original contents of the location being charged, and it displays the bars **I I** around the location being changed. The code used for this command is shown below.

Figure 8.17 *SYS5 Code From 4FDB to 500F*

```
*
*                          M Command
*
4FDB 2A6040      LD      HL,(4060H)  Current modify address
4FDE CDA351      CALL    51A3H       Get next modify address
4FE1 226040      LD      (4060H),HL  Save previous, or new addr if entered
4FE4 D8          RET     C           Ret to caller if carriage return
4FE5 E5          PUSH    HL          Save addr of location to be modified
4FE6 CDCF4E      CALL    4ECFH       Update display in case it's changed
4FE9 21403F      LD      HL,3F40H    Screen addr for current mem contents
4FEC 222040      LD      (4020H),HL  Save as cursor addr in video DCB
4FEF 2A6040      LD      HL,(4060H)  Modify address to HL
4FF2 CDD451      CALL    51D4H       Display address to be modified
4FF5 E5          PUSH    HL          Save addr of location to modify
4FF6 21803F      LD      HL,3F80H    Get addr to display modified mem and
4FF9 222040      LD      (4020H),HL  save as new cursor addr in video DCB
4FFC E1          POP     HL          Restore addr of location to modify
4FFD CDD051      CALL    51D0H       Display conts. of location to be
                                     modified. Follow it with a hyphen "-"
5000 3E2D        LD      A,2DH       ASCII value for -
5002 CD3300      CALL    0033H       Display -
5005 D1          POP     DE          Reload addr of location to modify
5006 CDA351      CALL    51A3H       Input new value for this location
5009 EB          EX      DE,HL       Save it in DE. Storage addr to HL
500A 801         JR      Z,500DH     If space input don't modify,
500C 73          LD      (HL),E      save new value
500D D8          RET     C           Exit if carriage return entered
500E 23          INC     HL          else bump to next address
500F 18D0        JR      4FE1H       Go update display, get replacement val.
```

All of the support routines used by this command have been discussed in previous sections.

8.6 'R' Command

This command is similar to the 'M' command except that it modifies a register's contents rather than those of a memory location. It has as a parameter a two or three character register designation. Two characters are used to signify a change in the main register set, while a three character name (two letter register characters plus a ') are used to indicate a change to the alternate register set.

The register specification is validated using the text list used for the register display. Any register changed is modified in the context save area. Following is the code used for this command:

Figure 8.18 *SYS5 Code From 5011 to 505C*

```
*
*                          R Command
*
5011 CD8A51      CALL    518AH       Get 1st letter of reg pair (next char)
5014 C8          RET     Z           Ret if comma, blank or C.R. (error)
5015 4F          LD      C,A         Save 1st letter of register pair
5016 CD8A51      CALL    518AH       Get 2nd letter of reg pair (next char)
5019 C8          RET     Z           Ret if comma, blank or C.R. (error)
501A 57          LD      D,A         Save 2nd letter of register pair
501B 1E20        LD      E,20H       Default RP terminator. Use for comma
501D CD8A51      CALL    518AH       Get char. after register pair.
                                     Should be blank or ","
5020 D8          RET     C           Ret if control. Error if not blank
5021 2806        JR      Z,5029H     Go if comma/blank (legal separator)
5023 5F          LD      E,A         Save actual char. Could be '
5024 CD8A51      CALL    518AH       Get next char. Must be comma/blank
5027 C0          RET     NZ          Exit if
5028 D8          RET     C           neither. Ignore illegal command
5029 21544F      LD      HL,4F54H    Address of register pair table
502C 060C        LD      B,0CH       Number of pairs in table. (12)
502E 7E          LD      A,(HL)      .   Get first char. of register pair
502F B9          CP      C           .   Compare to first char.
                                     .   of pair entered
5030 2806        JR      Z,5038H     .   If so, go test 2nd char
5032 23          INC     HL          .   Else skip over 2nd
5033 23          INC     HL          .   and 3rd char to 1st
5034 23          INC     HL          .   char of next entry
5035 10F7        DJNZ    502EH       .   Loop till all entries checked
5037 C9          RET                 .   Ignore cmd if no match in list
5038 23          INC     HL          .   Bump to 2nd char of RP name
5039 7E          LD      A,(HL)      .   Get 2nd char in RP name list
503A BA          CP      D           .   Compare to 2nd char.
                                     .   of name entered
503B 20F6        JR      NZ,5033H    .   Go if no match. Scan rest of list
503D 23          INC     HL          .   Bump to 3rd char in RP name list
503E 7E          LD      A,(HL)      .   Get it. Will be blank or quote
503F BB          CP      E           .   Compare to value entered
5040 20F2        JR      NZ,5034H    .   Go if no match (input error)
5042 3E18        LD      A,18H       2 times Max no. of entries in RP list
5044 90          SUB     B           Minus offset to
5045 90          SUB     B           matching entry gives
5046 4F          LD      C,A         byte offset into
5047 0600        LD      B,00H       saved register pair area
5049 216540      LD      HL,4065H    Origin of save register pair
                                     values in low memory
504C 09          ADD     HL,BC       Compute addr of RP to be modified
504D E5          PUSH    HL          Save addr of RP to be altered
504E 3E1E        LD      A,1EH       Code to erase to end of line
5050 CD3300      CALL    0033H       Blank remainder of line
5053 D1          POP     DE          DE = address of RP to change
5054 CDA351      CALL    51A3H       Put value for register pair in DE
5057 C8          RET     Z           Exit if illegal character entered
```

Listing Continued . . .

. . . Continued Listing

```
5058 EB         EX      DE,HL       Value to DE. Address to HL
5059 73         LD      (HL),E      Save LSB of new register value
505A 23         INC     HL          Bump to next byte
505B 72         LD      (HL),D      Save MSB of new register value
505C C9         RET                 Return to caller
```

8.7 'I,' 'C' Commands (Single Step)

These commands allow a program to be executed instruction by instruction under user control. The difference between the 'I' and 'C' commands is that when the 'C' command is used, all subroutine calls are executed in full with the TRAP occuring on return from the subroutine. When an 'I' command is used, the trap occurs at the next instruction executed.

Most of the code for the 'I' and 'C' commands is common. In addition to sharing code between themselves, code is shared with the 'G' command.

When instructions are single-stepped, a trap instruction replaces the instruction following the one to be stepped. The current instruction is then executed as though a 'G' command has been executed. When the instruction cycle completes and the following instruction is executed, DEBUG is re-entered because of the trap instruction.

In order to properly place the trap instruction, two things must be known. First, the byte length of the instruction to be executed must be found so the address of the following instruction can be computed. And second, the type of instruction to be executed must be known because control may not return in-line. Each instruction to be stepped is compared with up to three lists in order to determine its length and type.

Instructions are divided into six types. Type 0 instructions are non-branching, and non-direct. The next instruction to be executed immediately follows a type 0 instruction. Type 1 instructions indirectly branch through the address in the HL register. After execution, they do not return to the following instruction. Type 2 instructions are indirect jumps through the IX or IY registers. Type 3 instructions are long jumps. Type 4 instructions are short jumps such as JR or DJNZ. Type 5 instructions are RET type of instructions, and Type 6 are CALL's. Call instructions are identified repeatedly, so the 'I' or 'C' command selected can treat them accordingly.

Branch instructions which will not return to the next in-line instruction cause a TRAP to be stored at the target jump address. In addition, a TRAP is stored at the address for the following instruction in case the branch is conditional. This means the status flags need not be tested for conditional jumps since a trap will be placed at both possible exits.

Following is the code used for the 'I' and 'C' commands:

Figure 8.19 *SYS5 Code From 505D to 50DE*

```
*
*                Single Step (I and C Commands)
*
505D F5          PUSH    AF           Save command specification
505E ED5B7B40    LD      DE,(407BH)   Load resumption address
5062 1A          LD      A,(DE)       Load OP code at 'resume' address
5063 211551      LD      HL,5115H     Addr of group 3 type instructions
5066 FEDD        CP      0DDH         Test for indexed instruction
5068 280E        JR      Z,5078H      Jump if indexed IX instr.
506A FEFD        CP      0FDH         If not, test for IY indexed instr.
506C 280A        JR      Z,5078H      Jump if indexed IY
506E 21E050      LD      HL,50E0H     Addr of group 1 type instructions
5071 FEED        CP      0EDH         Test for special instruction
5073 2006        JR      NZ,507BH     Go if not indexed or special instr.
5075 210E51      LD      HL,510EH     Addr of group 2 type instructions
5078 13          INC     DE           Bump to next byte of instruction
5079 1A          LD      A,(DE)       Fetch next instruction byte
507A 1B          DEC     DE           Backspace to start of instruction
507B 4F          LD      C,A          C = LSB of OP code
507C 7E          LD      A,(HL)       .   Fetch OP code mask
507D A1          AND     C            .   Apply mask to LSB of OP code
507E 23          INC     HL           .   Bump to OP code value in
                                      .   instruction list
507F BE          CP      (HL)         .   Compare to OP code in table
5080 23          INC     HL           .   Bump to code length in list
5081 2806        JR      Z,5089H      .   Jump if OP codes match, else
5083 23          INC     HL           .   bump to next 3 byte entry
5084 7E          LD      A,(HL)       .   Fetch mask
5085 FE05        CP      05H          .   and test for end of list
5087 30F3        JR      NC,507CH     .   Go if end of list not reached
5089 7E          LD      A,(HL)       Get code/length value for instruction
508A 47          LD      B,A          Save in B
508B E60F        AND     0FH          Isolate length of instruction to
                                      be single cycled
508D 6F          LD      L,A          Then form length of instruction
508E 2600        LD      H,00H        in HL, and
5090 19          ADD     HL,DE        add to addr of instr. to execute
5091 D5          PUSH    DE           Save address of next instruction
5092 116240      LD      DE,4062H     Load trap address save location
5095 CDCA4F      CALL    4FCAH        Original instr to 4062. Trap to (HL)
5098 E1          POP     HL           Restore addr of instr after current
5099 78          LD      A,B          Reload code/length for current instr
509A E6F0        AND     0F0H         Isolate type e.g. RET, JP, CALL ...
509C 2816        JR      Z,50B4H      Go if type 0. Exec. cont. at next addr.
509E 23          INC     HL           Bump addr. of next instr. by one
509F FE20        CP      20H          Test for JP type instr.
50A1 3835        JR      C,50D8H      Jump if JP (HL) instr. (Type 1)
50A3 2827        JR      Z,50CCH      Jump if JP (IX/IY) instr. (Type 2)
50A5 FE40        CP      40H          Test for JP  XXXX type instr.
```

Listing Continued . . .

. . . Continued Listing

```
50A7 3817       JR      C,50C0H     Jump if DJNZ (Type 3)
50A9 280F       JR      Z,50BAH     Jump if JP  XXXX (Type 4)
50AB FE60       CP      60H         Test for type 6
50AD 3808       JR      C,50B7H     Jump if code 5 (RET type instr.)
50AF F1         POP     AF          Restore command (I or C)
50B0 FE49       CP      49H         Compare with I. If so go
50B2 2806       JR      Z,50BAH     store trap at CALL addr and execute
50B4 C3A44F     JP      4FA4H       C command. Go restore context
                                    and execute instruction
50B7 2A7940     LD      HL,(4079H)  Load ret addr from context stack
50BA 7E         LD      A,(HL)      LSB of return addr.
50BB 23         INC     HL          Bump to MSB of ret.addr.
50BC 66         LD      H,(HL)      load MSB of return addr.
50BD 6F         LD      L,A         HL = return address
50BE 181B       JR      50DBH       Go restore context and exec. instr.
50C0 4E         LD      C,(HL)      Fetch offset for DJNZ, JR
50C1 AF         XOR     A           Clear status flags
50C2 CB79       BIT     07H,C       Test sign bit of offset displacement
50C4 2801       JR      Z,50C7H     Jump if positive. Forward jump
50C6 2F         CPL                 Negative displacement. Convert to
50C7 47         LD      B,A         8 bit negative value
50C8 23         INC     HL          Bump to addr of following instr.
50C9 09         ADD     HL,BC       Add offset to get addr of next instr.
50CA 180F       JR      50DBH       Go restore context, execute instr.
50CC 2A7540     LD      HL,(4075H)  Load IX in case of JP (IX)
50CF CB69       BIT     05H,C       Test if JP (IX)
50D1 2808       JR      Z,50DBH     If JP (IX) go execute instruction
50D3 2A7740     LD      HL,(4077H)  Load IY. JP (IY) instr.
50D6 1803       JR      50DBH       Go restore context, execute instr.
50D8 2A6B40     LD      HL,(406BH)  Get jump address from HL
50DB CDCA4F     CALL    4FCAH       Stuff trap instruction there
50DE 18D4       JR      50B4H       Go restore context, execute instr.
```

8.8 'G' Command

The 'G' command causes the TRAP'd program to be restarted. An execution address may be specified as an option; otherwise, execution resumes at the PC address in the saved context.

Up to two TRAP addresses may also be optionally specified. If TRAP addresses are given, the original contents of the locations specified by the TRAP addresses are saved in the trap list table beginning at **4062**.

Before passing control to the resumed program, all context at the time of the trap is restored. Control is returned via an RET instruction as it requires no registers. The following code is used for 'G' command processing:

Figure 8.20 *SYS5 Code From 4F80 to 4FDA*

```
*
*                              G Command
*
4F80 0602       LD      B,02H         Max. no. trap addr to input
4F82 116240     LD      DE,4062H      Address of trap table
4F85 CDA351     CALL    51A3H         Input a hex. value (1st address)
4F88 2803       JR      Z,4F8DH       Jump if illegal char. detected
4F8A 227B40     LD      (407BH),HL    Save new PC (continue address)
4F8D 380A       JR      C,4F99H       .   Jump if no comma after 1st address
4F8F CDA351     CALL    51A3H         .   Get a hex.value (2nd address)
4F92 F5         PUSH    AF            .   Preserve status
4F93 C4CA4F     CALL    NZ,4FCAH      .   Call routine to update trap
                                      .   table addr. if legit. value
4F96 F1         POP     AF            .   Restore status 1st addr read
4F97 10F4       DJNZ    4F8DH         .   Loop till 2 addr or car. ret found
4F99 210F40     LD      HL,400FH      Vector address to load DEBUG
4F9C 221643     LD      (4316H),HL    put in nucleus
4F9F 3EC3       LD      A,0C3H        OP code for JUMP instruction
4FA1 321543     LD      (4315H),A     put in nucleus
4FA4 217A40     LD      HL,407AH      Addr of context starting with SP
4FA7 060B       LD      B,0BH         No. of reg. context values to transfer
*
*     Transfer register values from context area to current stack.
*
4FA9 56         LD      D,(HL)        .   Fetch MSB of saved value
4FAA 2B         DEC     HL            .   Skip back to LSB
4FAB 5E         LD      E,(HL)        .   Fetch LSB of saved register value
4FAC 2B         DEC     HL            .   Skip to next register value
4FAD D5         PUSH    DE            .   Save value on current stack
4FAE 10F9       DJNZ    4FA9H         .   Loop till all values on stack
*
*                Restore Register Context
*
4FB0 F1         POP     AF            Restore AF main register
4FB1 C1         POP     BC            Restore BC main register
4FB2 D1         POP     DE            Restore DE main register
4FB3 E1         POP     HL            Restore HL main register
4FB4 08         EX      AF,AF'        Switch to alternate AF register
4FB5 D9         EXX                   Switch to alternate register set
4FB6 F1         POP     AF            Restore AF alternate register
4FB7 C1         POP     BC            Restore BC alternate register
4FB8 D1         POP     DE            Restore DE alternate register
4FB9 E1         POP     HL            Restore HL alternate register
4FBA 08         EX      AF,AF'        Switch to main AF register
4FBB D9         EXX                   Switch to main register
4FBC DDE1       POP     IX            Restore IX
4FBE FDE1       POP     IY            and IY
```

Listing Continued . . .

. . . Continued Listing

```
4FC0 E1          POP     HL          and stack pointer at trap time
4FC1 F9          LD      SP,HL       Update SP to value at time of trap
4FC2 2A7B40      LD      HL,(407BH)  Resumption address to HL
4FC5 E5          PUSH    HL          So it can be copied to stack
4FC6 2A6B40      LD      HL,(406BH)  Restore HL main register set
4FC9 C9          RET                 Return to inst. where trap occurred
*
*    Stuff trap inst. at addr. in HL. Save original contents and
*    addr. of contents at list addr. in DE.
*
4FCA 7E          LD      A,(HL)      Get original conts. of trap location.
4FCB 12          LD      (DE),A      and save in trap table list
4FCC 1B          DEC     DE          Back-up to next entry in trap list
4FCD 3EF7        LD      A,0F7H      Load a trap inst. RST 30
4FCF 77          LD      (HL),A      Save at addr. in HL, then
4FD0 BE          CP      (HL)        Make sure store worked
4FD1 C22F4E      JP      NZ,4E2FH    Jump if store inst. failed
4FD4 7C          LD      A,H         MSB of trap addr.
4FD5 12          LD      (DE),A      To trap table
4FD6 1B          DEC     DE          Backspace trap table addr.
4FF7 7D          LD      A,L         LSB of trap addr.
4FD8 12          LD      (DE),A      To trap table
4FD9 1B          DEC     DE          Back-up to 1st byte of next entry
4FDA C9          RET                 Return to caller
```

8.9 'U' Command

The 'U' command indirectly enables interrupts, allowing clock-driven functions to execute. As soon as any key is struck, the 'U' command exits to the main DEBUG loop. Following is the code used for this command:

Figure 8.21 *SYS5 Code From 4E9E to 43A6*

```
*
*              U Command. Dynamic Display
*
4E9E CD2B00      CALL    002BH       .   Scan keyboard
4EA1 B7          OR      A           .   Set flags for active key test
4EA2 C0          RET     NZ          .   Ret to 4E4B if any key active
4EA3 CDCF4E      CALL    4ECFH       .   Update display
4EA6 18F6        JR      4E9EH       .   Loop till any key struck
```

notes

9

9.0 SYS6/SYS

SYS6 is the overlay module containing code for the bulk of the TRSDOS commands. The following commands are processed by SYS6:

APPEND ATTRIB AUTO CLOCK COPY
DATE DEVICE DIR DUMP FREE
KILL LIB LIST LOAD PRINT
PROT RENAME TIME VERIFY

SYS6 is different from the other overlay modules in several repects. First, it is called only from SYS1 when one of the commands listed above is detected.

Second, SYS6 loads at **5200** rather than **4E00**. This fact would prohibit SYS6 from being called by a processor utility program such as BACKUP or FORMAT since they have the same load address.

A third distinction of SYS6 is its direct references to subroutines in the underlying overlay. Since SYS6 is called by SYS1, it makes use of subroutines in SYS1 by direct CALL statements. Subroutines in SYS2 are also used. When they are needed, system calls (usually an OPEN) are made, causing SYS2 to be loaded into the overlay area.

A fourth difference is that SYS6 expects the HL register to contain the beginning address of the remainder of the command string upon entry. Parsing of the command string parameters will be done by SYS1, SYS2 or SYS6 depending on the command. All of the parsing routines used will expect HL to contain the address of the string to be parsed.

The final difference between SYS6 and its cousins regards the passing of parameters. All other overlays, except for SYS5, expect a parameter to be passed as part of the overlay request code which is left in the A register. Bits 4, 5, and 6 of the overlay request are reserved for parameter passing. SYS6 expects a parameter in the C register. This parameter is a value between 1 and 19 which corresponds to the command to be executed.

Structurally, SYS6 is quite simple. A small amount of code at the beginning interrogates the parameter by decrementing it and jumping to a fixed address when a zero is detected. The fixed address defines the beginning of the code for each of the command functions. If the value of the parameter exceeds 19, the message RESERVED COMMAND is displayed, and control returns to SYS1 via the nucleus call at **4030**. The following code is used to test the parameter and branch to the command function:

Figure 9.1 *SYS6 Code From 5200 to 5257*

```
5200 79        LD      A,C          Get command index computed by SYS1
5201 012D40    LD      BC,402DH     Put a return address of 402D
5204 C5        PUSH    BC           Onto the stack
5205 3D        DEC     A            Test if command index = 1
5206 CA0057    JP      Z,5700H      Jump if APPEND command
5209 3D        DEC     A            Test if command index = 2
520A CA005B    JP      Z,5B00H      Jump if ATTRIB command
520D 3D        DEC     A            Test if command index = 3
520E CA8352    JP      Z,5283H      Jump if AUTO command
5211 3D        DEC     A            Test if command index = 4
5212 CAC552    JP      Z,52C5H      Jump if CLOCK command
5215 3D        DEC     A            Test of command index = 5
5216 CA4F57    JP      Z,574FH      Jump if COPY command
5219 3D        DEC     A            Test if command index = 6
521A CA9D52    JP      Z,529DH      Jump if DATE command
521D 3D        DEC     A            Test if command index = 7
521E CA1D53    JP      Z,531DH      Jump if DEVICE command
5221 3D        DEC     A            Test if command index = 8
5222 CA6F5C    JP      Z,5C6FH      Jump if DIR command
5225 3D        DEC     A            Test if command index = 9
5226 CAD7557   JP      Z,57D7H      Jump if DUMP command
5229 3D        DEC     A            Test if command index = 10
522A CA205E    JP      Z,5E20H      Jump if FREE command
522D 3D        DEC     A            Test if command index = 11
522E CA1459    JP      Z,5914H      Jump if KILL command
5231 3D        DEC     A            Test if command index = 12
5232 CA3A53    JP      Z,533AH      Jump if LIB command
5235 3D        DEC     A            Test if command index = 13
5236 CA2C59    JP      Z,592CH      Jump if LIST command
5239 3D        DEC     A            Test if command index = 14
523A CA9459    JP      Z,5994H      Jump if LOAD command
523D 3D        DEC     A            Test if command index = 15
523E CAAC59    JP      Z,59ACH      Jump if PRINT command
5241 3D        DEC     A            Test if command index = 16
5242 CA6A53    JP      Z,536AH      Jump if PROT command
5245 3D        DEC     A            Test if command index = 17
5246 CA9C5F    JP      Z,5F9CH      Jump if RENAME command
5249 3D        DEC     A            Test if command index = 18
524A CAB152    JP      Z,52B1H      Jump if TIME command
524D 3D        DEC     A            Test if command index = 19
524E CA0455    JP      Z,5504H      Jump if VERIFY command
*
```

Listing Continued . . .

. . . Continued Listing

```
*               Command Index Out of Range
*
5251 215A52     LD      HL,525AH      Addr of RESERVED COMMAND message
5254 CD6744     CALL    4467H         Display message, then
5257 C33040     JP      4030H         Recall SYS1, option 2
```

All command functions return control to SYS1 through a nucleus call to **4020** or **4030**. In some error cases, control will pass to SYS4 before final control is returned to SYS1.

9.1 APPEND Command

The APPEND command is an undocumented TRSDOS feature. At least, my TRSDOS manual doesn't mention it. The APPEND command combines two files. Its parameters are two file names. The APPEND command skips to the end of file of the second file and copies the first file onto the second file. The APPEND command is called using the following command:

APPEND Filename1 Filename2

Code for the APPEND function consists of two sections. The first begins at **5700** and ends at **574C**. The second begins at **57B6** and ends at **57D5**.

The second section is entered directly from the first section by a direct jump. The second section determines when processing is complete and jumps into the tail end of the COPY copy to close the files and return control to SYS1.

There is no explanation for breaking APPEND into two sections. It may have been added as an afterthought. In that case, the code would have had to occupy whatever space was available.

Section one parses the command by calling SYS1 to copy the file names into local DCB areas at **5551** and **5571** (herein referred to as file one and file two, respectively). SYS1 will check the file names for illegal characters and return a non-zero status if one is found. If an error is encountered, the appropriate error message is displayed, and the operation is aborted.

Following the calls to SYS1 to copy the file names to the DCB's, the files are OPEN'ed. Before the files were OPEN'ed, the names should have been compared to insure they were not the same, or at least on different drives.

After the files have been OPEN'ed, a nucleus subroutine at **4448** is called to skip to end of file on file two. When end of file has been reached on file two, control passes to Section Two. Following is the code for Section One:

Figure 9.2 *SYS6 Code From 5700 to 574C*

```
*
*               APPEND command processing
*
5700 115155     LD      DE,5551H      Address of local DCB area
5703 CD1C44     CALL    441CH         Copy/edit file into DCB
                                      (SYS1, code 4)
```

Listing Continued . . .

. . . Continued Listing

```
5706 C23A55      JP       NZ,553AH      Display FILE SPEC. REQUIRED
                                        (error if no file name)
5709 1A          LD       A,(DE)        Fetch 1st character of file name
570A FE2A        CP       2AH           Test for reference to special file
570C CA3A55      JP       Z,553AH       Display FILE SPEC REQUIRED
                                        (special file reference)
570F 117155      LD       DE,5571H      Address of 2nd local DCB area
5712 CD1C44      CALL     441CH         Copy 2nd file name to DCB area
5715 C23A55      JP       NZ,553AH      Display FILE SPEC REQUIRED
                                        (error in 2nd file name)
5718 1A          LD       A,(DE)        Get first char of 2nd file name
5719 FE2A        CP       2AH           Test for reference to special file
571B CA3A55      JP       Z,553AH       Display FILE SPEC REQUIRED
                                        (special file reference)
571E 0600        LD       B,00H         Specify physical I/O
5720 210056      LD       HL,5600H      Sector buffer addr for 2nd file
5723 CD2444      CALL     4424H         OPEN 2nd file
5726 C24C55      JP       NZ,554CH      Jump if any error during OPEN
5729 0600        LD       B,00H         Specify physical I/O
572B 115155      LD       DE,5551H      DCB address of 1st file
572E 21006F      LD       HL,6F00H      Sector buffer addr for first file
5731 CD2444      CALL     4424H         OPEN second file
5734 C24C55      JP       NZ,554CH      Go if error while OPENing 2nd file
5737 117155      LD       DE,5571H      DCB address for 2nd file
573A CD4844      CALL     4448H         Skip to end of file on 2nd file
573D 00          NOP
573E 00          NOP
573F 00          NOP
5740 00          NOP
5741 00          NOP
5742 00          NOP
5743 00          NOP
5744 00          NOP
5745 00          NOP
5746 00          NOP
5747 00          NOP
5748 00          NOP
5749 00          NOP
574A 00          NOP
574B 00          NOP
574C C3B657      JP       57B6H         Copy file 1 onto end of file 2
```

Section Two reads file one, one character at a time, writing each character onto file two. The code for this operation is very brief. It terminates when end-of-file is reached on file one, or any other error condition is detected in file one or file two.

After end-of-file is reached on file one, both files are CLOSE'd, and control returns to SYS1. Following is the code for Section Two:

Figure 9.3 *SYS6 Code From 57B6 to 57D5*

```
57B6 115155     LD      DE,5551H     DCB address for first file
*
*          Continuation of APPEND Processing
*
57B9 CD1300     CALL    0013H        Read a character from lst file
57BC 2807       JR      Z,57C5H      Jump if no error condition
57BE FE1C       CP      1CH          An error occurred. Was it end of file
57C0 28DF       JR      Z,57A1H      Jump if yes (EOF or source file
57C2 C34C55     JP      554CH        If not, Call SYS4 to process error
57C5 47         LD      B,A          Preserve character just read
57C6 CD1555     CALL    5515H        Test for BREAK/SHIFT@
57C9 20D6       JR      NZ,57A1H     If BREAK key active
                                     terminate operation
57CB 78         LD      A,B          Restore character read
57CC 117155     LD      DE,5571H     DCB addr of 2nd file (dest. file)
57CF CD1B00     CALL    001BH        Write char from source to dest. file
57D2 C24C55     JP      NZ,554CH     Jump if error during write operation
57D5 18DF       JR      57B6H        Else loop till EOF on source file
```

9.2 ATTRIB Command

The ATTRIB command is used to assign or modify file password and protection levels. The format of the command is:

ATTRIB filename (options)

where options can be one of the following:

I	Assign the invisible attribute to the file. This means the file name will not be displayed during DIR commands. In order to view the file name, I option must be used with the DIR command.
ACC = password	Assign new access password.
UPD = password	Assign new update password.
PROT = level	Assign an access protection level, assignable levels are KILL, RENAME, WRITE, READ, and EXEC.

The code for ATTRIB processing begins at **5B00** and ends at **5C3F**. A local DCB at **5551** is used to OPEN the file.

The code begins with a call to SYS1 with an option code of 40. This copies the file name from the command line buffer to the local DCB area. Upon return from SYS1, the file is OPEN'ed. Opening the file serves a dual purpose in that it causes SYS2 to be loaded into the system overlay area as well as initializing the DCB. This fact is important since subroutines in SYS2 may be called to compute the encode of ACCESS and UPDATE passwords.

After opening the file, the access permission flags in the DCB are isolated and compared to zero. If non-zero, the file is being accessed without the proper permission, and an error message is generated via a call to SYS4.

Assuming no access violation occurred, two local variables at **5C05** and **5COD** are zeroed. These locations will be modified by subsequent processing to contain the file visibility flag and the access permission flags.

Next, the command line is scanned until a '(' is detected. The following code is used for these operations:

Figure 9.4 *SYS6 Code From 5B00 to 5B38*

```
*
*               Attribute Command Processing
*
5B00 115155      LD      DE,5551H     Local DCB address
5B03 CD1C44      CALL    441CH        SYS1, code 4. File name to DCB
5B06 C23A55      JP      NZ,553AH     Jump if error detected in file name
5B09 1A          LD      A,(DE)       Is first character
5B0A FE2A        CP      2AH          of file an *
5B0C CA3A55      JP      Z,553AH      If so, display FILE SPEC. REQUIRED
5B0F E5          PUSH    HL           Save addr of parameters
5B10 0600        LD      B,00H        Specify physical I/O
5B12 21004D      LD      HL,4D00H     Sector buffer address
5B15 CD2444      CALL    4424H        OPEN file (load SYS2 at 4E00)
5B18 C24C55      JP      NZ,554CH     Jump if file does not exist
5B1B E1          POP     HL           Restore command string address
5B1C 3A5255      LD      A,(5552H)    Fetch permission flags for file
5B1F E607        AND     07H          Isolate access flags
5B21 3E37        LD      A,37H        Signal restricted access
                                      Invisible file, occupied entry
5B23 C24C55      JP      NZ,554CH     Jump if any permission required
5B26 AF          XOR     A            Load a zero and
5B27 32055C      LD      (5C05H),A    clear values to be combined with
5B2A 320D5C      LD      (5C0DH),A    access and invisibility attributes
5B2D 7E          LD      A,(HL)       Get next char from command string
5B2E FE28        CP      28H          Test for (
5B30 2808        JR      Z,5B3AH      Jump if (. Start of parameters
5B32 FE20        CP      20H          Not (. Test for blank
5B34 C26B52      JP      NZ,526BH     Not ( or , so ignore command
5B37 23          INC     HL           Blank. Skip to next character
5B38 18F3        JR      5B2DH        Loop till ( or end of command found
```

When a '(' has been located, the beginning of the options field has been found. The character following the '(' is compared to an: 'I' (INVISIBLE), 'A' (ACCESS), 'U' (UPDATE) or 'P' (PROTECT). When a match is found with one of these characters, control passes to a section of code which processes the parameter associated with the option. The following code is used to determine which of the options is present:

Figure 9.5 *SYS6 Code From 5B3A to 5B4A*

```
5B3A 23          INC     HL              Bump past (
5B3B 7E          LD      A,(HL)          Get next char of parameter field
5B3C FE49        CP      49H             Test for I
5B3E 2857        JR      Z,5B97H         Jump if INVISIBLE specified
5B40 FE41        CP      41H             Not I, test for ACC
5B42 2863        JR      Z,5BA7H         Jump if A found. Go test for CC
5B44 FE55        CP      55H             Not I or A. Test for UPD
5B46 2874        JR      Z,5BBCH         Jump if U. Go test for PD
5B48 FE50        CP      50H             Not I, A, or U. Test for PROT
5B4A C2395C      JP      NZ,5C39H        None of above. Go display
                                         ATTRIBUTE error message
```

If the PROT option has been selected, further processing is required to evaluate the keyword following the equals. The code below is used to process the PROT option.

Figure 9.6 *SYS6 Code From 5B4D to 5B94*

```
5B4D CDD15B      CALL    5BD1H           Parse PROT command  parameter
5B50 CA395C      JP      Z,5C39H         Error if not (= KILL/RENAME/WRITE
                                         WRITE/READ or PROT
5B53 E5          PUSH    HL              Save command string address
5B54 0607        LD      B,07H           Number of entries to compare
5B56 ED5B5551    LD      DE,(5155H)      Get 1st 2 chars of PROT parameter
5B5A 21615C      LD      HL,5C61H        List of possible parameters
5B5D 7E          LD      A,(HL)          .   Get one character from list
5B5E 23          INC     HL              .   Bump to next char in list
5B5F BB          CP      E               .   Compare char of parameter to list
5B60 CC695B      CALL    Z,5B69H         .   If equal, try matching others
5B63 23          INC     HL              .   else bump to next option in list
5B64 10F7        DJNZ    5B5DH           Loop till all options compared
5B66 C3395C      JP      5C39H           Error, no match w/PROT options
*
*                Test PROT parameters
*
5B69 7E          LD      A,(HL)          Fetch 2nd char from option test
5B6A BA          CP      D               Compare with 2nd char of PROT option
5B6B C0          RET     NZ              Exit if not equal. Try next option
5B6C F1          POP     AF              Clear ret addr from stack. Will
                                         ret to 5BF7/5B3A or 5C39
5B6D 78          LD      A,B             No. of options remaining in list
```

Listing Continued . . .

. . . Continued Listing

```
5B6E 3D          DEC     A              Decrement for matching option
5B6F FE05        CP      05H            Are we on RE (READ or RENAME)
5B71 200B        JR      NZ,5B7EH       Jump if not RE
5B73 3A5751      LD      A,(5157H)      Fetch 3rd char of PROT parameter.
                                        Test for RENAME
5B76 FE4E        CP      4EH            Test for N (RENAME not READ
5B78 3E05        LD      A,05H          Access flags for READ
5B7A 2002        JR      NZ,5B7EH       Jump if READ option
5B7C 3E02        LD      A,02H          Access flags for RENAME
5B7E 320B5C      LD      (5C0BH),A      Save access permission for PROT
5B81 E1          POP     HL             Restore command string address
5B82 0601        LD      B,01H          KILL/RENAME/READ/WRITE/EXECUTE access
5B84 3A055C      LD      A,(5C05H)      Byte 0 of dir entry w/access field 0
5B87 B0          OR      B              Merge with new access permission
5B88 32055C      LD      (5C05H),A      Save updated byte to update dir
5B8B 7E          LD      A,(HL)         Get next char from command string
5B8C FE29        CP      29H            Test for )
5B8E 2867        JR      Z,5BF7H        Jump if ) found (end of command)
5B90 FE2C        CP      2CH            No left paren. Test for comma
5B92 28A6        JR      Z,5B3AH        Go if , found (more parameters follow)
5B94 C3395C      JP      5C39H          else error in attribute spec.
```

A subroutine common to all of the options processing is shown below:

Figure 9.7 *SYS6 Code From 5BD1 to 5BF6*

```
*
*                    Parse PROT Parameters
*
5BD1 23          INC     HL             .   Bump to next addr in command
5BD2 7E          LD      A,(HL)         .   Fetch next character
5BD3 FE0D        CP      0DH            .   Test for end of line
5BD5 C8          RET     Z              .   Exit if end of line
5BD6 FE29        CP      29H            .   Test for closing paren
5BD8 C8          RET     Z              .   Exit if )
5BD9 FE2C        CP      2CH            .   Test for comma
5BDB C8          RET     Z              .   Exit if comma
5BDC FE3D        CP      3DH            Test for =
5BDE 20F1        JR      NZ,5BD1H       Not = , loop
5BE0 23          INC     HL             Bump to 1st char following =
5BE1 115551      LD      DE,5155H       Address of buffer area in SYS2
5BE4 0608        LD      B,08H          No. of buffer bytes to initialize
5BE6 D5          PUSH    DE             Save beginning buffer address
5BE7 C5          PUSH    BC             Save max no. of bytes in buffer
5BE8 3E20        LD      A,20H          Initialization value
5BEA 12          LD      (DE),A         .   Initialize a buffer byte
5BEB 13          INC     DE             .   Bump to next buffer byte
5BEC 10FC        DJNZ    5BEAH          .   Loop till buffer initialized
5BEE C1          POP     BC             Restore buffer size value in B
5BEF D1          POP     DE             Restore beginning buffer address
5BF0 CD8150      CALL    5081H          Copy password to buffer address
                                        in DE. Use subroutine in SYS2
```

Listing Continued . . .

. . . Continued Listing

```
5BF3 2B          DEC     HL          Back-up cmd $ pointer to terminator
5BF4 F601        OR      01H         Signal password found
5BF6 C9          RET                 and return to caller
```

Following is the code for the 'I' option:

Figure 9.8 *SYS6 Code From 5B97 to 5BA5*

```
*
*                    ATTRIB - I Processing
*
5B97 CDD15B      CALL    5BD1H       Parse rest of command
5B9A C2395C      JP      NZ,5C39H    Jump if command string contains
5B9D 3A00D5C     LD      A,(5C00DH)  more than ). Get access permission
5BA0 F608        OR      08H         from dir entry. Set invisible flag
5BA2 3200D5C     LD      (5C00DH),A  Save updated access permission flags
5BA5 18DD        JR      5B84H       Go test for closing ) and write
                                     updated directory entry
```

Following is the code for the ACC option:

Figure 9.9 *SYS6 Code From 5BA7 to 5BBA*

```
*
*                    ATTRIB - ACC Processing
*
5BA7 CDD15B      CALL    5BD1H       Parse rest of command
5BAA CA395C      JP      Z,5C39H     Error if nothing follows A
5BAD E5          PUSH    HL          Save current cmd string address
5BAE 115551      LD      DE,5155H    Address of ACCESS password
5BB1 CDD150      CALL    50D1H       Get encode of ACCESS password
5BB4 226A51      LD      (516AH),HL  Save encode
5BB7 E1          POP     HL          Restore command string address
5BB8 0602        LD      B,02H       RENAME/WRITE/READ/EXECUTE access
5BBA 18C8        JR      5B84H       Go update access permission
                                     field in directory
```

Following is the code for the UPD option:

Figure 9.10 *SYS6 Code From 5BBC to 5BCF*

```
*
*                    ATTRIB - UPD Processing
*
5BBC CDD15B      CALL    5BD1H       Parse rest of command
5BBF CA395C      JP      Z,5C39H     Jump if nothing follows U
5BC2 E5          PUSH    HL          Save command string address
```

Listing Continued . . .

. . . Continued Listing

```
5BC3 115551      LD      DE,5155H      Address of buffer UPDATE password
5BC6 CDD150      CALL    50D1H         Get encode of UPDATE password
                                       (use subroutine in SYS2)
5BC9 226851      LD      (5168H),HL    Save encode
5BCC E1          POP     HL            Restore command string address
5BCD 0604        LD      B,04H         Specify WRITE/READ/EXECUTE access
5BCF 18B3        JR      5B84H         Go update access permission
                                       in directory sector
```

Notice that all of the option processing routines return to **5B84**, except for the PROT option which falls into this code. At this point, the access flags are generated and saved at **5C05**. Next, the command line is scanned for a ')' or a comma. If a comma is found, control returns to **5B3A**, where the next option is processed.

When a ')' is found, control is passed to **5BF7**, where the directory sector for the file is read, updated and rewritten. Following is the code for those operations.

Figure 9.11 *SYS6 Code From 5BF7 to 5C36*

```
5BF7 ED4B5755    LD      BC,(5557H)    Get dir sector pointer from DCB
5BFB CDC14A      CALL    4AC1H         Read dir sector into buffer
5BFE C24C55      JP      NZ,554CH      Jump if error during read
5C01 7E          LD      A,(HL)        Fetch 1st byte of dir entry
5C02 E6F8        AND     0F8H          Clear access permissions
5C04 1600        LD      D,00H         Load new access permission
5C06 CB42        BIT     00H,D         Test if all permission granted
5C08 2802        JR      Z,5C0CH       Jump if not
5C0A F600        OR      00H           Combine access permission bits
5C0C F600        OR      00H           With file visability attribute
5C0E 77          LD      (HL),A        Save updated access in dir entry
5C0F 7D          LD      A,L           Skip to UPDATE access password
5C10 C610        ADD     A,10H         by adding 10 to current dir address
5C12 6F          LD      L,A           for address of UPDATE password
5C13 CB52        BIT     02H,D         Test if UPDATE password specified
5C15 280A        JR      Z,5C21H       Jump if not
5C17 3A6851      LD      A,(5168H)     Yes. Get new UPDATE password encode
5C1A 77          LD      (HL),A        and save in directory
5C1B 3A6951      LD      A,(5169H)     Fetch 2nd byte of encode
5C1E 23          INC     HL            Bump to 2nd byte of password
5C1F 77          LD      (HL),A        Save 2nd byte of UPDATE encode
5C20 2B          DEC     HL            Back-up for following increments
5C21 23          INC     HL            Bump to starting position of access
5C22 23          INC     HL            password in directory
5C23 CB4A        BIT     01H,D         Test for KILL/RENAME/READ
                                       WRITE/EXECUTE access
5C25 2809        JR      Z,5C30H       Jump if all options allowed
5C27 3A6A51      LD      A,(516AH)     Else get UPDATE password encode (LSB)
5C2A 77          LD      (HL),A        and copy to directory
5C2B 3A6B51      LD      A,(516BH)     Fetch UPDATE password encode (MSB)
5C2E 23          INC     HL            Bump to 2nd byte of password
5C2F 77          LD      (HL),A        Store MSB encode in directory
```

Listing Continued . . .

. . . Continued Listing

```
5C30 CDD64A     CALL    4AD6H       Write updated directory sector
5C33 C24C55     JP      NZ,554CH    Jump if any error during WRITE
5C36 C32D40     JP      402DH       Done. Recall SYS1, wait for command
```

9.3 AUTO Command

The AUTO command copies the next 20 characters from the command line into the AUTO procedure area of the GAT sector (track 11, sector 1). The method used is quite simple.

First, the GAT sector is read into a buffer at **4D00** using a subroutine in the nucleus. Next, 20 characters are copied from the current position in the command line buffer to **4DE0** in the GAT sector buffer. Finally, the GAT sector is rewritten using another nucleus subroutine. All disk operations are restricted to drive 0.

Following is the code for the AUTO command:

Figure 9.12 *SYS6 Code From 5282 to 529C*

```
*
*                   AUTO Command Processing
*
5283 0E00       LD      C,00H       Specify drive 0
5285 CDF04A     CALL    4AF0H       Get track 11, sector 0 into 4D00
5288 C24C55     JP      NZ,554CH    Jump if error while reading GAT
528B 164D       LD      D,4DH       Form addr of AUTO procedure in
528B 1EE0       LD      E,0E0H      GAT sector buffer in DE (4DE0)
528F 012000     LD      BC,0020H    No. of chars to put in AUTO area
5292 EDB0       LDIR                Copy 32 chars following AUTO command
                                    to procedure file name field
5294 0E00       LD      C,00H       Select drive 0
5296 CD034B     CALL    4B03H       Write updated GAT sector
5299 C24C55     JP      NZ,554CH    Go if error during write
529C C9         RET                 Return to SYS1
```

9.4 CLOCK Command

Code for the CLOCK command begins at **52C5**. It starts by calling a subroutine in SYS1 to process the ON/OFF portion of the CLOCK command. If the Word OFF is detected, the status register is set to zero before returning.

Upon return from SYS1, the address of the clock maintenance routine in the nucleus is loaded into the DE register. This is necessary if the command is turning the clock on. The A register is loaded with a 6. This is the index (divided by 2) into the interrupt service routine address list for the clock maintenance routine. Next, depending on the status register returned from SYS1, control goes to **4410** or **4413**, where the clock address is added to, or deleted from the interrupt service routine list depending on the option selected.

Control will pass from the routine at **4410** and **4413** to SYS1 when they complete because **402D** was pushed onto the stack when SYS1 was entered.

Following is the code for the CLOCK command:

Figure 9.13 *Code from 52C5 to 52D0*

```
*
*                    CLOCK Routine
*
52C5 CDA051      CALL    51A0H        SYS1 ON/OFF parameter processor
52C8 11A94C      LD      DE,4CA9H     Clock maintenance routine address
52CB 3E06        LD      A,06H        Index for clock routine in
                                      interrupt service list
52CD CA1344      JP      Z,4413H      Remove addr from list, go to SYS1
52D0 C31044      JP      4410H        Add addr to list. Go to SYS1
```

9.5 COPY Command

The COPY command begins at address **574F**. It starts with a call to SYS1, option 40 to copy the source file name to a DCB area at **5551**. A second option 40 call to SYS1 is made to copy the destination file name to another DCB area at **5571**.

The option 40 subroutine compares the file name being copied to the phrases TO, ON and OVER. If it matches any of them, the command line is rescanned under the assumption that a separator was detected. Obviously, this could be a false assumption. It means that file names of TO, ON and OVER cannot be used as destination files on a COPY command.

After the source and destination file names have been copied to their DCB's, the source file is OPEN'ed, and the destination file is opened with an INIT call. Both files are opened for physical I/O, and they share a sector buffer at **5600**. The following code is used for the operation described.

Figure 9.14 *SYS6 Code From 574F to 577A*

```
*
*                 COPY Command Processing
*
574F 115155      LD      DE,5551H     Address of local DCB area
5752 CD1C44      CALL    441CH        SYS1, code 4, copy/edit 1st file
5755 C23A55      JP      NZ,553AH     Go if illegal chars in file name
5758 117155      LD      DE,5571H     Address of 2nd local DCB area
575B CD1C44      CALL    441CH        SYS1, code 4, copy/edit 2nd file name
575E C23A55      JP      NZ,553AH     Jump if illegal chars 2nd file name
5761 0600        LD      B,00H        Set physical I/O for source file
5763 115155      LD      DE,5551H     Address of source DCB
5766 210056      LD      HL,5600H     Sector buffer address
```

Listing Continued . . .

. . . Continued Listing

```
5769 CD2444     CALL    4424H       OPEN source file
576C C24C55     JP      NZ,554CH    Jump if any error during OPEN
576F 0600       LD      B,00H       Set physical I/O for destination file
5771 117155     LD      DE,5571H    Destination DCB address
5774 210056     LD      HL,5600H    Share sector buf with source file
5777 CD2044     CALL    4420H       INIT call for destination file
577A C24C55     JP      NZ,554CH    Jump if error during INIT processing
```

After the files have been opened, the copy operation commences. The source file is read, a sector at a time, and written to the destination file. Reading continues until an end of file, or record not found status is returned. If the reading terminates because of a record not found status, the last record accessed is copied from the source DCB to the destination DCB before closing the files.

When the read terminates, both files are closed, and control returns to SYS1 via a call to the nucleus entry point at **402D**. Following is the code used for these operations:

Figure 9.15 *SYS6 Code From 577D to 57B3*

```
577D 115155     LD      DE,5551H    .   Address of source DCB
5780 CD3644     CALL    4436H       .   Read sector from source file
5783 280B       JR      Z,5790H     .   Jump if no error during read
5785 FE1C       CP      1CH         .   Test for EOF on source file
5787 2818       JR      Z,57A1H     .   Jump if EOF on source
5789 FE1D       CP      1DH         .   else test for record not found
578B 280E       JR      Z,579BH     .   Jump if record not found
578D C34C55     JP      554CH       .   Some other error. Call SYS4
                                    .   to display error message
5790 117155     LD      DE,5571H    .   No error on source file.
                                    .   Load address of dest. DCB
5793 CD3944     CALL    4439H       .   Write sector just read to
                                    .   destination file
5796 28E5       JR      Z,577DH     .   Go if no error during WRITE
5798 C34C55     JP      554CH       Loop till EOF on source file
579B 2A5955     LD      HL,(5559H)  Record not found. Get record no.
579E 227955     LD      (5579H),HL  from source DCB, copy to dest. DCB
57A1 117155     LD      DE,5571H    Address of destination DCB
57A4 CD2844     CALL    4428H       CLOSE destination file
57A7 C24C55     JP      NZ,554CH    Jump if any error during CLOSE
57AA 115155     LD      DE,5551H    Address of source DCB
57AD CD2844     CALL    4428H       CLOSE source file
57B0 C24C55     JP      NZ,554CH    Jump if error during CLOSE
57B3 C32D40     JP      402DH       Return to SYS1 wait for next command
```

9.6 DATE Command

The DATE command copies the date from the command list to the date buffer in the system area. Before the date can be moved, it must be converted from ASCII to binary. A subroutine at **52EA** is called for this conversion.

After the date has been converted, it is copied to the date buffer in the system area. The following code is used for procesing the DATE command.

Figure 9.16 *SYS6 Code From 529D to 52B0*

```
*
*                       DATE Routine
*
529D 0E2F       LD      C,2FH       / = Only valid terminating character
529F CDEA52     CALL    52EAH       Process date. Leave in 52DC-52DE
52A2 C2D352     JP      NZ,52D3H    Go if char not a number or /
52A5 21DC52     LD      HL,52DCH    Start of 3 byte date buffer
52A8 114440     LD      DE,4044H    Address of date in system area
52AB 010300     LD      BC,0003H    Number of bytes to move
52AE EDB0       LDIR                Copy date from buffer to system area
52B0 C9         RET                 Ret to SYS1 to get next command
```

The subroutine at **52EA** converts a string of ASCII digits to binary and stores the results in a local buffer which begins at **52DE**. Since this subroutine is called to process time and date strings, the separator (either a ':' or '/') between values must be specified at the time the subroutine is called.

One subroutine is called from **52EA** to convert two digits from ASCII to binary. It begins at **52FF** and immediately calls a subroutine at **5316** to return the hex value for the next character. This value is multiplied by 10, and the subroutine at **5316** is called again to fetch the next digit, which is added to the value computed for the first digit. This gives the decimal-binary equivalent of a two digit ASCII number.

Control returns to the subroutine at **52EA** with the binary value for the two digit number in the A register. Since only dates and times are expected to be processed, the A register has sufficient bit width to hold any valid numbers.

The value returned is stored in a local buffer, and the character following the number is compared to the separator in the C register. If they match, the next two digit number is processed and saved. This procedure is repeated until three sets of two digit numbers have been converted.

If a non-numeric character or an illegal separator is detected, control returns to the caller with a non-zero status; otherwise, a zero status is returned. The following code is used for these operations:

Figure 9.17 *SYS6 Code From 52EA to 531C*

```
52EA 11DE52     LD      DE,52DEH    Ending addr of local 3 byte buffer
52ED 0603       LD      B,03H       Number of fields to process
52EF D5         PUSH    DE          Save storage address
52F0 CDFF52     CALL    52FFH       Process next field. Must be 2 digits
52F3 D1         POP     DE          Restore buffer address
52F4 C0         RET     NZ          Exit if error in field.
                                    Non-numeric encountered
52F5 12         LD      (DE),A      Save a date field
52F6 1B         DEC     DE          Backup to next storage address
52F7 05         DEC     B           Count one field processed
52F8 C8         RET     Z           Exit if all fields processed
52F9 7E         LD      A,(HL)      Get field separator from command
52FA 23         INC     HL          Bump to 1st char of next field
52FB B9         CP      C           Compare separator to specified value
52FC 28F1       JR      Z,52EFH     Loop if legitimate separator
52FE C9         RET                 Ret to caller if illegal separator
*
*
*
52FF CD1653     CALL    5316H       Get next char from command string
5302 3010       JR      NC,5314H    Jump if alpha character (error)
5304 5F         LD      E,A         Save value (will be added later)
5305 07         RLCA                Value * 2
5306 07         RLCA                Value * 4
5307 83         ADD     A,E         Value * 5
5308 07         RLCA                Value * 10
5309 5F         LD      E,A         Save binary equivalent of digit
530A CD1653     CALL    5316H       Get next char from command string
530D 3005       JR      NC,5314H    Jump if not numeric (error)
530F 83         ADD     A,E         Combine with value of previous
5310 5F         LD      E,A         Digit * 10
5311 AF         XOR     A           Set status flag for no error
5312 7B         LD      A,E         Binary equivalent of decimal
                                    value in A-reg.
5313 C9         RET                 Return to caller
5314 B7         OR      A           Signal error with non-zero status
5315 C9         RET                 Ret. Alpha character encountered
*
*
*
5316 7E         LD      A,(HL)      Fetch real char from command buffer
5317 23         INC     HL          Bump to next character
5318 D630       SUB     30H         Convert ASCII digit to binary
531A FE0A       CP      0AH         Test for non-numeric character
531C C9         RET                 Ret w/CARRY if char greater than 9
```

9.7 DEVICE Command

The DEVICE command displays the device mnemonics maintained in the system device table. The following code is used for this command:

Figure 9.18 *SYS6 Code From 531D to 5337*

```
*
*                       DEVICE Routine
*
531D 21C043     LD      HL,43C0H      Device table address
5320 7E         LD      A,(HL)        .   Get ASCII char of device type
5321 2C         INC     L             .   Bump to next char in device name
5322 B7         OR      A             .   Check for end of list
5323 2812       JR      Z,5337H       .   Jump if end of device table, else
5325 CD3300     CALL    0033H         .   display 1st char of device type
5328 7E         LD      A,(HL)        .   Get 2nd char of device type
5329 CD3300     CALL    0033H         .   Display it then
532C 3E0D       LD      A,0DH         .   follow with car. ret. to
532E CD3300     CALL    0033H         .   skip to next line
5331 7D         LD      A,L           .   LSB of device table address
5332 C603       ADD     A,03H         .   Add 3 to get next entry.
                                      .   Skip over driver address
5334 6F         LD      L,A           .   Reform addr of next entry in HL
5335 30E9       JR      NC,5320H      .   Test if absolute end of
                                      .   list reached (43CF)
5337 C32D40     JP      402DH         .   Return to SYS1
```

9.8 DIR Command

The DIR command is one of the longest commands in SYS6. The command has four options. They are:

DIR
DIR (I)
DIR (S)
DIR (A)

The 'A,' 'I' and 'S' options may be combined, as long as they are seperated by commas. A drive may also be specified.

After testing for a drive parameter, SYS1 (option 60) is called to parse the command for the 'I,' 'S' and 'A' options. SYS1 will examine the balance of the DIR command. If any of the options are present, a non-zero value will be stored at the following address:

```
A - 5D9B/C
I - 5D0B/C
S - 5CFD/E
```

The contents of these addresses are initialized to zero at the start of DIR processing. Later they will be tested (acting as program switches), and branches will be made (if they are non-zero) to sections of code containing the instructions required to produce the display for the option selected.

The code for DIR begins at **5C6F**. It starts by clearing the aforementioned program switches, displaying the DIR title and displaying the diskette name and creation date. The following code is used for these operations:

Figure 9.19 *SYS6 Code From 5C6F to 5CC4*

```
*
*                    DIR command processing
*
5C6F 010000      LD      BC,0000H     Load 16-Bit zero value to clear
5C72 ED439B5D    LD      (5D9BH),BC   A option flag (set not selected)
5C76 ED430B5D    LD      (5D0BH),BC   I option flag (set not selected)
5C7A ED43FD5C    LD      (5CFDH),BC   S option flag (set not selected)
5C7E 0E00        LD      C,00H        Default drive number is 0
5C80 7E          LD      A,(HL)       Get next char from command list
5C81 FE3A        CP      3AH          Compare with a :
5C83 2006        JR      NZ,5C8BH     Jump if no drive specified
5C85 23          INC     HL           Drive specified. Bump cmd list addr
5C86 7E          LD      A,(HL)       to drive. Get ASCII drive no.
5C87 23          INC     HL           Bump to character beyond drive no.
5C88 D630        SUB     30H          Convert drive number to binary
5C8A 4F          LD      C,A          and save in C for nucleus I/O calls
5C8B C5          PUSH    BC           Save drive for later use
5C8C 79          LD      A,C          Refetch it
5C8D C630        ADD     A,30H        and convert it to ASCII
5C8F 329E5E      LD      (5E9EH),A    Save drive no. in title message
5C92 116A5E      LD      DE,5E6AH     Address of DIR option I, S, A
5C95 CD7644      CALL    4476H        Call SYS1 to parse rest of command
5C98 21835E      LD      HL,5E83H     Address of DIR title line
5C9B CD6744      CALL    4467H        Display title line
5C9E C1          POP     BC           Reload drive number in C
5C9F C5          PUSH    BC           And preserve BC
5CA0 CD554B      CALL    4B55H        Get dir track number in D register
5CA3 1E00        LD      E,00H        Select sector 0
5CA5 210042      LD      HL,4200H     Use system buffer at 4200
5CA8 CD354B      CALL    4B35H        Read track 11, sector 0
5CAB C24C55      JP      NZ,554CH     Jump if error during read
5CAE 21D042      LD      HL,42D0H     Addr of disk name in GAT sec buf
5CB1 11A15E      LD      DE,5EA1H     Addr of disk name in 3rd title line
5CB4 010800      LD      BC,0008H     Number of bytes in diskette name
5CB7 EDB0        LDIR                 Move name from GAT buffer to 3rd line
5CB9 11AD5E      LD      DE,5EADH     Addr of date in 3rd title line
5CBC 010800      LD      BC,0008H     Number of bytes in date
5CBF EDB0        LDIR                 Move date to 3rd title line
5CC1 21A15E      LD      HL,5EA1H     Address of 3rd title line
5CC4 CD6744      CALL    4467H        Display 3rd title line
```

Next, all eight directory sectors are read and stored in a buffer starting at **6100**. The code used for this is shown below:

Figure 9.20 *SYS6 Code From 5CC7 to 5CE0*

```
5CC7 C1        POP     BC          Restore drive no. to C
5CC8 0600      LD      B,00H       Setup for reading 1st dir sector
5CCA 110061    LD      DE,6100H    Sector buffer address
5CCD CDC14A    CALL    4AC1H       Read sector of dir file into 4200
5CD0 C24C55    JP      NZ,554CH    Jump if error during read
5CD3 2E00      LD      L,00H       Force HL to 4200. Sector buf origin
5CD5 C5        PUSH    BC          Save sector number/drive number
5CD6 010001    LD      BC,0100H    Number of bytes to move (256)
5CD9 EDB0      LDIR                Copy buffer at 4200 to 6100
5CDB C1        POP     BC          Restore sector/drive number
5CDC 04        INC     B           Bump sector number
5CDD 78        LD      A,B         Move sector number for comparison
5CDE FE08      CP      08H         Did we read last directory sector
5CE0 20EB      JR      NZ,5CCDH    Jump if not. Read next sector
```

After all directory sectors have been read, the display loop is entered. This loop consists of three identifiable sections.

The first begins at **5CE5** and ends at **5DOF**. This section of code determines if the current directory entry is eligible for display. The code for this section is shown below:

Figure 9.21 *SYS6 Code From 5CE5 to 5D0F*

```
5CE5 3E0C      LD      A,0CH       No. of lines to display before pause
5CE7 32AB5D    LD      (5DABH),A   Initialize line counter
5CEA 0603      LD      B,03H       No. of files displayed per line
5CEC 7E        LD      A((HL)      Fetch access control byte from dir
5CED E5        PUSH    HL          Save dir addr for current file
5CEE CB67      BIT     04H,A       Test if entry is occupied
5CF0 CABF5D    JP      Z,5DBFH     If not, go get next entry
5CF3 CB7F      BIT     07H,A       Entry used. Is it primary or overflow
5CF5 C2BF5D    JP      NZ,5DBFH    Jump if overflow. Skip this
                                   entry, get next one
5CF8 CB77      BIT     06H,A       Test if user or SYSTEM file
5CFA 280A      JR      Z,5D06H     Jump if SYSTEM file
5CFC 110000    LD      DE,0000H    FFFF if S option specified
5CFF 7A        LD      A,D         Test if user specified S option
5D00 B3        OR      E           DE = 0000 if not (FFFF if so)
5D01 CABF5D    JP      Z,5DBFH     Jump if S option not selected
5D04 180C      JR      5D12H       S option selected. Go display file
5D06 CB5F      BIT     03H,A       Test if file is invisible
5D08 2808      JR      Z,5D12H     Jump if file name is displayable
```

Listing Continued . . .

. . . Continued Listing

```
5D0A 110000      LD      DE,0000H     FFFF if I option specified
5D0D 7A          LD      A,D          DE = 0000 if not, FFFF otherwise
5D0E B3          OR      E            Contents of 5D0B-5D0C modified by
                                      SYS1 if I option detected
5D0F CABF5D      JP      Z,5DBFH      Jump if I option not selected
```

The second section displays the file name and (depending on the options selected) the file attributes. This section can be divided into three parts. The first part displays the file name and type. The following code is used:

Figure 9.22 *SYS6 Code From 5D12 to 5D64*

```
5D12 C5          PUSH    BC           Save file's line count/drive no.
5D13 7D          LD      A,L          LSB of current entry address
5D14 C605        ADD     A,05H        Add 5 to form addr of file name
5D16 6F          LD      L,A          HL = addr of name in dir sector
5D17 0E11        LD      C,11H        Max no. chars in file name/ext
5D19 0608        LD      B,08H        Max no. chars in file name
5D1B 7E          LD      A,(HL)       .  Fetch a character from name
5D1C 23          INC     HL           .  Bump to next character
5D1D FE20        CP      20H          .  Is last character a blank
5D1F 2808        JR      Z,5D29H      .  if yes, go display extension
5D21 CD3300      CALL    0033H        .  Display a char from file name
5D24 0D          DEC     C            .  Count 1 character displayed
5D25 10F4        DJNZ    5D1BH        .  Loop till name displayed
5D27 1804        JR      5D2DH        Name displayed. Go display ext
5D29 7D          LD      A,L          Name not 8 chars. Get LSB of current
5D2A 80          ADD     A,B          addr in dir buf. Add remainder of
5D2B 3D          DEC     A            bytes (-1) left in name to get
                                      address of extension
5D2C 6F          LD      L,A          Form address of extension in HL
5D2D 7E          LD      A,(HL)       Fetch 1st byte of extension
5D2E FE20        CP      20H          and compare it to a blank
5D30 2814        JR      Z,5D46H      If blank, no ext. Go blank ext area
5D32 3E2F        LD      A,2FH        ASCII /
5D34 CD3300      CALL    0033H        Display / after file name
5D37 0D          DEC     C            Dec count for total field size
5D38 0603        LD      B,03H        Max no. chars in extension
5D3A 7E          LD      A,(HL)       .  Fetch a byte from ext name
5D3B 23          INC     HL           .  Bump to next char in extension
5D3C FE20        CP      20H          .  Is current char a blank
5D3E 2806        JR      Z,5D46H      .  Jump if yes. End of extension
                                      .  reached. Go get next file
5D40 CD3300      CALL    0033H        .  Display current char in ext
5D43 0D          DEC     C            .  Dec count for total field size
5D44 10F4        DJNZ    5D3AH        .  Loop till extension displayed
5D46 3E20        LD      A,20H        Follow file name or extension
5D48 CD3300      CALL    0033H        With a blank
5D4B 0D          DEC     C            Dec count displayed in field
5D4C 7D          LD      A,L          Fetch LSB of current file entry
```

Listing Continued . . .

. . . Continued Listing

```
5D4D E6E0       AND     0E0H        Force LSB to beginning of file
5D4F 6F         LD      L,A         Form beginning of file addr in HL
5D50 46         LD      B,(HL)      Fetch access flag
5D51 3E53       LD      A,53H       ASCII S in case of system file
5D53 CB70       BIT     06H,B       Is it a system file
5D55 2002       JR      NZ,5D59H    Jump if yes, display S
5D57 3E20       LD      A,20H       No, replace S with a space
5D59 CD3300     CALL    0033H       Display space/S, depending on type
5D5C 3E49       LD      A,49H       ASCII I in case file is invisible
5D5E CB58       BIT     03H,B       Test if file viewable
5D60 2002       JR      NZ,5D64H    Jump if file invisible
5D62 3E20       LD      A,20H       Not invisible, replace I with space
5D64 CD3300     CALL    0033H       Display space/I,
                                    depending on attribute
```

The second part of section two determines if the file has a password, and if so, displays a 'P' after the file name. This part of section two is executed only if the 'E' or 'S' option has been selected. The following code is used to test for password protection on the file:

Figure 9.23 *SYS6 Code From 5D67 to 5D98*

```
5D67 E5         PUSH    HL          Save beginning addr of file entry
5D68 7D         LD      A,L         Get LSB of file entry address
5D69 C610       ADD     A,10H       Add 10 to get addr of UPDATE password
5D6B 6F         LD      L,A         Form addr of UPDATE password in HL
5D6C 5E         LD      E,(HL)      Fetch LSB of update password
5D6D 2C         INC     L           Bump to next byte in password
5D6E 56         LD      D,(HL)      Fetch MSB of update password
5D6F E5         PUSH    HL          Save address of UPDATE password
5D70 219642     LD      HL,4296H    Value for universal password
5D73 ED52       SBC     HL,DE       Test if file password protected
5D75 E1         POP     HL          Restore addr of UPDATE password
5D76 2814       JR      Z,5D8CH     Jump if file has no password
5D78 78         LD      A,B         Fetch access flags
5D79 E607       AND     07H         Isolate access control flags
5D7B 3E50       LD      A,50H       ASCII P
5D7D 200F       JR      NZ,5D8EH    Jump if not unrestricted access
5D7F 2C         INC     L           Bump to LSB of ACCESS password
5D80 5E         LD      E,(HL)      Fetch LSB of ACCESS password
5D81 2C         INC     L           Bump to MSB of ACCESS password
5D82 56         LD      D,(HL)      Fetch MSB of ACCESS password
5D83 219642     LD      HL,4296H    Universal password code
5D86 ED52       SBC     HL,DE       Is there any ACCESS password
5D88 3E50       LD      A,50H       ASCII P in case there is a password
5D8A 2002       JR      NZ,5D8EH    Go if ACCESS password. Display P
5D8C 3E20       LD      A,20H       No password. Replace P with space
5D8E CD3300     CALL    0033H       Disp. P/space,
                                    depending on permission
```

Listing Continued . . .

. . . Continued Listing

```
5D91 E1          POP     HL              Restore beginning addr of file entry
*
*                Blank fill remainder of field
*
5D92 3E20        LD      A,20H           .   A = ASCII space
5D94 CD3300      CALL    0033H           .   Display a space
5D97 0D          DEC     C               .   Dec count for total field size
5D98 20F8        JR      NZ,5D92H        .   Loop till field filled
```

The third part of section two is called only if the 'A' option has been selected. The code at **5D9A** tests for this option and also maintains the files per line count using the following instructions:

Figure 9.24 *SYS6 Code From 5D9A to 5DBD*

```
5D9A 110000      LD      DE,0000H        FFFF if A option specified
5D9D 7A          LD      A,D             Combine LSB and MSB
5D9E B3          OR      E               Of A option byte. Changed by SYS1
                                         if A option specified
5D9F C1          POP     BC              Restore files/line count, drive no.
5DA0 C4F15D      CALL    NZ,5DF1H        If A option, go display file size
5DA3 101A        DJNZ    5DBFH           Count 1 file displayed. Jump if
                                         room for more files this line
5DA5 3E0D        LD      A,0DH           Else skip to next line
5DA7 CD3300      CALL    0033H           By displaying a carriage return
5DAA 3E00        LD      A,00H           Get max no. of lines per screen
5DAC 3D          DEC     A               Decrement it
5DAD 2005        JR      NZ,5DB4H        Go if room for more lines this screen
5DAF CD4900      CALL    0049H           Else wait for operator reply
5DB2 3E0C        LD      A,0CH           Max. number of lines per screen
5DB4 32AB5D      LD      (5DABH),A       reset line/screen counter
5DB7 CD1555      CALL    5515H           Test for BREAK key
5DBA C2DB5D      JP      NZ,5DDBH        Jump if BREAK key active
5DBD 0603        LD      B,03H           Reload no. of files/line
counter
```

If the 'A' option has been selected, the following subroutine is called to display the logical record length and file size.

Figure 9.25 *SYS6 Code From 5DF1 to 5E65*

```
*
*       'A' Option of DIR Command. Display File Size.
*
5DF1 E5          PUSH    HL              Save beginning addr of file entry
5DF2 2C          INC     L               Bump by 3
5DF3 2C          INC     L               to get address of
5DF4 2C          INC     L               EOF sector
```

Listing Continued . . .

. . . Continued Listing

```
5DF5 7E          LD      A,(HL)        Fetch EOF byte offset
5DF6 32695E      LD      (5E69H),A     Save in temp. storage area
5DF9 2C          INC     L             Bump to logical record length in dir
5DFA E5          PUSH    HL            Save address of LRL
5DFB 6E          LD      L,(HL)        Fetch logical record length
5DFC 7D          LD      A,L           Move it to A so it can
5DFD 321F5E      LD      (5E1FH),A     be saved in temp. storage
5E00 D601        SUB     01H           Sub 1 to force CARRY if LRL = 256 (0)
5E02 3E00        LD      A,00H         Then add CARRY to a
5E04 CE00        ADC     A,00H         Byte of zeroes. This yields a
5E06 67          LD      H,A           16 Bit value for LRL (01-0100)
5E07 11BB5E      LD      DE,5EBBH      Addr of LRL in display line
5E0A CD6C60      CALL    606CH         Convert to ASCII, save in display
5E0D E1          POP     HL            Restore address of LRL
5E0E 7D          LD      A,L           Fetch LSB of address
5E0F C610        ADD     A,10H         and add 10 to addr of byte
                                       containing EOF sector no.
5E11 6F          LD      L,A           Reform addr of EOF sector in HL
5E12 7E          LD      A,(HL)        Fetch LSB of ending sector no.
5E13 32685E      LD      (5E68H),A     Save in temp. storage
5E16 2C          INC     L             Bump to MSB of EOF sector
5E17 7E          LD      A,(HL)        Fetch MSB of ending sector no.
5E18 32675E      LD      (5E67H),A     Save in temp. storage
5E1B 21675E      LD      HL,5E67H      Addr of ending sector no. in binary
5E1E 0E00        LD      C,00H         Load logical record length.
                                       Instr. modified at 5DFD
5E20 CDA760      CALL    60A7H         Divide no. of records by record size
5E23 EB          EX      DE,HL         Quotient to HL
5E24 11C75E      LD      DE,5EC7H      Addr of ending sec no. in display
5E27 CD6C60      CALL    606CH         Convert ending sector no. to ASCII
5E2A E1          POP     HL            Restore address of LRL
5E2B CD3E5E      CALL    5E3EH         Go compute no. grans assigned to file
5E2E EB          EX      DE,HL         Move granule count to HL
5E2F 11D45E      LD      DE,5ED4H      Addr of no. grans in display line
5E32 CD6C60      CALL    606CH         Convert gran count to decimal ASCII
5E35 21B75E      LD      HL,5EB7H      Address of LRL display line
5E38 CD6744      CALL    4467H         Display LRL/EOF offset and # grans
5E3B 0601        LD      B,01H         Replace line counter with 1 because
5E3D C9          RET                   only one file/line with this
                                       display. Return to caller
*
*
*
5E3E 110000      LD      DE,0000H      Zero accumulator
5E41 7D          LD      A,L           Fetch LSB of LRL
5E42 C616        ADD     A,16H         Add 16 to position to GAP's
5E44 6F          LD      L,A           Form address of GAP's in HL
5E45 7E          LD      A,(HL)        .   Fetch 1st byte of a GAP
5E46 2C          INC     L             .   Bump to granule count
5E47 FEFE        CP      0FEH          .   Test for end of GAP's or overflow
5E49 300C        JR      NC,5E57H      .   Jump if end or overflow
5E4B 7E          LD      A,(HL)        .   Load granule count
5E4C 2C          INC     L             .   Bump to next GAP
5E4D E61F        AND     1FH           .   Isolate granule count
5E4F 3C          INC     A             .   Compute true granule count
```

Listing Continued . . .

. . . Continued Listing

```
5E50 83          ADD     A,E          .   Form sum in DE
5E51 5F          LD      E,A          .   Update LSB of sum
5E52 30F1        JR      NC,5E45H     .   Jump if LSB has not oveflowed
5E54 14          INC     D            .   else bump MSB of gran count
5E55 18EE        JR      5E45H        .   Loop till end of GAP's found
5E57 C0          RET     NZ           Exit if FF found (end of
GAP's)
5E58 46          LD      B,(HL)       Get pointer to overflow entry
5E59 78          LD      A,B          Move to A
5E5A E607        AND     07H          and isolate the sector number
5E5C C661        ADD     A,61H        Then form addr of sector buffer
5E5E 67          LD      H,A          and move MSB of addr to H
5E5F 78          LD      A,B          Now compute offset within sector
5E60 E6E0        AND     0E0H         Isolate offset to file entry
5E62 C616        ADD     A,16H        and add offset to GAP's
5E64 6F          LD      L,A          Form addr of overflow GAP's in HL
5E65 18DE        JR      5E45H        Continue till end of GAP's found
```

The third section of DIR code computes the address of the next directory entry in the sector buffer. The first entry from each sector is processed, then the second, etc. The following code is used to compute the address of the next file entry.

Figure 9.26 *SYS6 Code From 5DBF to 5DF0*

```
5DBF E1          POP     HL           Restore beginning addr of
                                      current directory entry
5DC0 24          INC     H            Bump to next sector
5DC1 7C          LD      A,H          Test if end of sector list reached
*
*      Loop control for DIR. All directory sectors are read
*      into 8 sector buffers beginning at 6100, 6200, . . .
*      before the main loop is entered. Each pass through the
*      loop processes one file entry from a sector buffer. At
*      the end of the loop the MSB of the file entry address
*      is incremented by 100 giving the address of the
*      corresponding entry in the next sector. When the end
*      of the sector buffers is reached, the LSB of the file
*      entry address is incremented by 20 to get the address
*      of the next entry in the last sector. Then the MSB
*      byte of the next entry address is set to 61 giving the
*      address of the next file entry in the first sector,
*      etc.
*
5DC2 FE69        CP      69H          Have 8 sectors been processed?
5DC4 C2EC5C      JP      NZ,5CECH     If not go process entry from next sec
5DC7 2661        LD      H,61H        Reset MSB of buf pointer to 1st sec
5DC9 7D          LD      A,L          Fetch LSB of beginning address
                                      for last entry
5DCA C620        ADD     A,20H        Add 20 to compute beginning addr
5DCC 6F          LD      L,A          of next entry. form addr in HL
```

Listing Continued . . .

. . . Continued Listing

```
5DCD D2EC5C     JP      NC,5CECH    Jump if all entries not processed
5DD0 CDE15D     CALL    5DE1H       Zero sector buffers
                                    to protect passwords
5DD3 3E0D       LD      A,0DH       Skip to next line
5DD5 CD3300     CALL    0033H       By displaying a carriage return
5DD8 C32D40     JP      402DH       Ret to SYS1 to get next command
5DDB CDE15D     CALL    5DE1H       Zero sector buffer for security
5DDE C33040     JP      4030H       Ret to SYS1 for next command
*
*
*
5DE1 210061     LD      HL,6100H    Beginning addr of sector buffer
5DE4 0600       LD      B,00H       Sector index
5DE6 70         LD      (HL),B      Zero a byte of a sector
5DE7 2C         INC     L           Bump to next byte of sector
5DE8 20FC       JR      NZ,5DE6H    Loop till one sector zeroed
5DEA 24         INC     H           Bump to next sector address
5DEB 7C         LD      A,H         Test if all buffers cleared
5DEC FE69       CP      69H         End of sector buffers reached
5DEE 20F6       JR      NZ,5DE6H    Jump if not. Zero next sector
5DF0 C9         RET                 Yes, return to caller
```

9.8 DUMP Command

The DUMP command writes a specified portion of memory to a disk file. The disk file contains embedded control bytes recognized by the loader. Thus, the file can be loaded and executed the same as binary files created by the editor/assembler.

There are two sections to the DUMP command. First is the initialization section. This section parses the DUMP command, opens the output file, writes the file name onto the file, and verifies the START, END, and TRA parameters.

There are four steps in the initialization process. First, local variables are initialized, next the parameters from the DUMP command are processed. Third, a six-character file name which will be written to the file is prepared, the file is opened, and the file name is written with loader control codes. Fourth, the parameters are verified; START must be less than END and START must be greater than **7000**:

Code for the first step begins at **5707** and is shown below.

Figure 9.27 *SYS6 Code From 57D7 to 57E5*

```
*
*               DUMP Command Processing
*
57D7 010070     LD      BC,7000H    Default START address DUMP
57DA ED436858   LD      (5868H),BC  Initialize START address location
57DE ED436E58   LD      (586EH),BC  Initialize END address location
57E2 012D40     LD      BC,402DH    Default TRANSFER address
57E5 ED43AB58   LD      (58ABH),BC  Initialize TRANSFER addr location
```

The locations initialized by these instructions (**5868**, **586E** and **58AB**) may be re-intialized by the SYS1 call at **57FB** if START, END and TRA parameters are present. If present, the hex values following those parameters will be stored at the address initialized in step one.

Step Two begins at **57E9** and ends at **57FB**; the following is used:

Figure 9.28 *SYS6 Code From 57E9 to 57FB*

```
57E9 115155      LD      DE,5551H     Address of local DCB area
57EC CD1C44      CALL    441CH        SYS1, code 4, copy/edit file name
57EF C23A55      JP      NZ,553AH     Jump if illegal chars in name
57F2 1A          LD      A,(DE)       None, check for
57F3 FE2A        CP      2AH          reference to special file (*)
57F5 CA3A55      JP      Z,553AH      Jump if found (error)
57F8 11F558      LD      DE,58F5H     Address of START text
57FB CD7644      CALL    4476H        SYS1, code 6. Look for word START
```

The call to SYS1 at **57EC** copies the file name to the local DCB. The call to SYS1 at **57FB** parses the remainder of the *DUMP* command looking for the START, END and TRA phrases.

Following is the code used for Step Three:

Figure 9.29 *SYS6 Code From 57FE to 584C*

```
57FE 110E59      LD      DE,590EH     Address of local buffer area
5801 215155      LD      HL,5551H     Address of local DCB
*
*     Copy 1st 6 characters of file name to 590E. Copy
*     terminates when a special character is detected or
*     6 characters have been copied
*
5804 0606        LD      B,06H        Max no. of chars to copy
5806 7E          LD      A,(HL)       .   Get a char from file name
5807 FE30        CP      30H          .   Test for special character
                                      .   (end of file name)
5809 3813        JR      C,581EH      .   Go if end of file name found
580B FE3A        CP      3AH          .   Test for numeric 0-9
580D 3808        JR      C,5817H      .   Jump if 0-9. Copy character
580F FE41        CP      41H          .   Test for special char 3A-3F
5811 380B        JR      C,581EH      .   Go if drive no. found. End copy
5813 FE5B        CP      5BH          .   Test for upper case letters
5815 3007        JR      NC,581EH     .   Go if upper case found. End copy
5817 12          LD      (DE),A       .   Char in A-Z, 0-9 range. Copy to
5818 23          INC     HL           .   Short list (590E) then bump
5819 13          INC     DE           .   Fetch and store address
```

Listing Continued . . .

. . . Continued Listing

```
581A 10EA       DJNZ    5806H       .   Loop till 6 chars copied
                                    .   or end of name found
581C 1806       JR      5824H       .   6 chars copied. Skip blank file
581E 3E20       LD      A,20H       .   ASCII blank to fill short list
5820 12         LD      (DE),A      .   Add to short list
5821 13         INC     DE          .   Bump to next pos. in short list
5822 10FA       DJNZ    581EH       .   Loop till 6 char name blanked
5824 115155     LD      DE,5551H    .   Addr of DCB for DUMP file
5827 21F258     LD      HL,58F2H    Address CIM text
582A CD7344     CALL    4473H       SYS1/code 5. Add CIM extension
582D 0600       LD      B,00H       Specify physical I/O
582F 210056     LD      HL,56       sector buffer address
5832 CD2044     CALL    4420H       INIT file
5835 C24C55     JP      NZ,554CH    Jump if error during INIT call
5838 3E05       LD      A,05H       Control code for file name
583A CD1B00     CALL    001BH       Send control code to disk file
583D 3E06       LD      A,06H       Number of chars in file name
583F CD1B00     CALL    001BH       Send byte count to disk file
5842 0606       LD      B,06H       Count for writing file name
5844 210E59     LD      HL,590EH    Address of file name buffer
5847 7E         LD      A,(HL)      Fetch a character from file name
5848 23         INC     HL          Bump to next character in name
5849 CD1B00     CALL    001BH       Send file name char to disk file
584C 10F9       DJNZ    5847H       Loop till name written (6 chars)
```

The code from **57FE** through **5822** copies six characters of the file name to a local buffer. The code is somewhat involved because the file name may be less than six characters. The purpose of the copy is to construct a six-character file name which will be written onto the file.

After the file name has been copied, a suffix of CIM (Core Image Memory) is added to the file name in the DCB. A SYS1 subroutine is used for this operation.

Next the file is opened, and the control bytes for a six byte text string are written to the disk file. These control bytes are followed by the six-character file name created earlier.

Finally, the fourth step of the initialization process is reached. This code verifies the START and END addresses. Error messages are produced if either are found to be out of range. The following code is used for this step:

Figure 9.30 *SYS6 Code From 584E to 5865*

```
584E 2A6E58     LD      HL,(586EH)  Load END addr given in DUMP command
5851 ED4B6858   LD      BC,(5868H)  Load START addr given in DUMP command
5855 AF         XOR     A           Clear carry
5856 ED42       SBC     HL,BC       END addr minus start address
5858 21C458     LD      HL,58C4H    Addr END LESS THAN START message
585B 3861       JR      C,58BEH     Jump if END less than START (error)
585D 21FF6F     LD      HL,6FFFH    Check for START address over 7000
5860 ED42       SBC     HL,BC       Compute 7000 minus START address
```

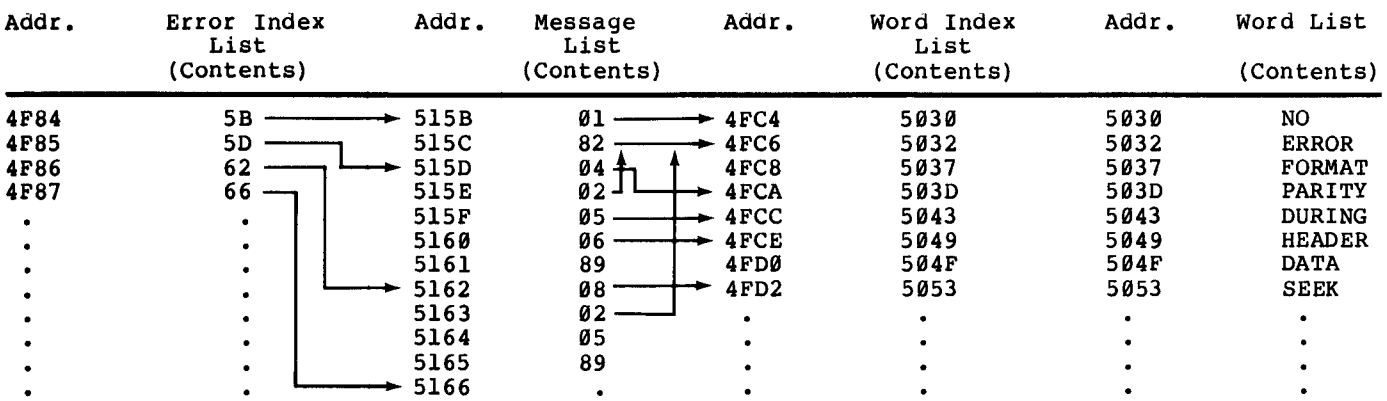
Listing Continued . . .

. . . Continued Listing

```
5862 21D958      LD      HL,58D9H     Addr START LESS THAN 7000 error msg.
5865 3057        JR      NC,58BEH     Go if START less than 7000 (error)
```

The code which writes the core image to disk has two special conditions to contend with. First, the core image must be written in blocks of 254 bytes, and second, each block must be preceded by loader control codes. The loader control codes consist of a control byte (**01** for binary data), a byte count of the number of bytes following the control code, a two-byte load address and the binary data to be loaded. A block of binary data would appear as follows:

Figure 9.31 *SYS4 Data Structure*



The end of a load module is signalled by control byte **02**. This control byte is followed by the beginning execution address for the module (also known as the transfer address).

The code which writes a binary load module can be divided into two sections. The first section contains the loop which writes the binary blocks. This loop computes the size of the block to be written, writes the control byte, byte count, load address and the binary data. The loop is repeated with the load address updated after each iteration until the last block of memory has been written. Following is the code used for this loop.

Figure 9.32 *SYS6 Code From 5867 to 589D*

```
5867 210000      LD      HL,0000H     START address. Filled in by SYS1
                                      from call at 57FB
586A E5          PUSH    HL           .   Save current START address
586B 44          LD      B,H          .   Then copy it
586C 4D          LD      C,L          .   To BC
586D 210000      LD      HL,0000H     .   END address. Filled in by
                                      .   SYS1 from call at 57FB
5870 23          INC     HL           .   END addr +1 for true byte count
5871 AF          XOR     A            .   Clear CARRY
5872 ED42        SBC     HL,BC        .   Compute no. of bytes to write
5874 2829        JR      Z,589FH      .   Go if done. Else break transfer
                                      .   into blocks of 254 bytes
5876 06FE        LD      B,0FEH       .   Max bytes between control bytes
5878 7C          LD      A,H          .   Test byte count for transfer
5879 B7          OR      A            .   Get status for upper 8 bits
587A 2006        JR      NZ,5882H     .   Go if transfer count > 255
587C 7D          LD      A,L          .   Transfer not > 255
587D FEFF        CP      0FFH         .   Check if = 255
587F 3001        JR      NC,5882H     .   Jump if it is. Use 254
5881 45          LD      B,L          .   Move count < 255. Use true val
5882 E1          POP     HL           .   Restore begin. addr current block
5883 3E01        LD      A,01H        .   Code 01 (load data follows)
5885 CD1B00      CALL    001Bh        .   Write control code 01
5888 78          LD      A,B          .   Get count. Add 2 for overhead
5889 C602        ADD     A,02H        .   If it was 254 (max), may write
588B CD1B00      CALL    001BH        .   a zero. Write count for DATA
588E 7D          LD      A,L          .   Then write LSB
588F CD1B00      CALL    001BH        .   of load address
5892 7C          LD      A,H          .   followed by MSB
5893 CD1B00      CALL    001BH        .   of load address
5896 7E          LD      A,(HL)       .   Then begin writing core image
5897 23          INC     HL           .   Bump to next fetch address
5898 CD1B00      CALL    001BH        .   Write a byte of core image
589B 10F9        DJNZ    5896H        .   Loop till block written
589D 18CB        JR      586AH        .   Go test if all blocks written
```

The second section writes the transfer address and closes the disk file. The transfer address consists of a control byte followed by the beginning execution address for the module. Following is the code used for this section:

Figure 9.33 *SYS6 Code From 589F to 58C1*

```
589F E1          POP     HL              .  Done, clear stack, prepare to
58A0 3E02        LD      A,02H           write transfer address. First
58A2 CD1B00      CAL1    001BH           Write a control code 02
58A5 3E02        LD      A,02H           followed by a byte count
58A7 CD1B00      CALL    001BH           for the address (2 of course)
58AA 210000      1D      HL,0000H        Transfer address filled in by
                                         SYS1 from call at 57FB
58AD 7D          Ld      A,L             Fetch a LSB of transfer address
58AE CD1B00      CALL    001BH           Write LSB of transfer address
58B1 7C          LD      A,H             Fetch MSB of transfer address
58B2 CD1B00      CALL    001BH           Write MSB of transfer address
58B5 CD2844      CALL    4428H           CLOSE disk file.
58B8 C24C55      JP      NZ,554CH        Jump if any error during CLOSE
58BB C32D40      JP      402DH           No error jump to SYS1
58BE CD6744      CALL    4467H           Write DUMP error message
58C1 C33040      JP      4030H           Ret to SYS1. Wait for next command
```

9.9 FREE Command

The FREE command dislays the diskette name and creation date, the number of available granules and the number of available HIT indices for each diskette currently mounted. The FREE command has no parameters.

Processing of the FREE command can be divided into five steps. Step one determines if the drive is ready; step two reads the GAT sector and moves the date and diskette name to the display line; step three computes the number of free granules; step four computes the number of available HIT entries; and step five dislays all of the above information. The following code is used for step one:

Figure 9.34 *SYS6 Code From 5EE0 to 5EF4*

```
*
*                    FREE Command Processing
*
5EE0 0E00        LD      C,00H           Default beginning drive number
5EE2 3E03        LD      A,03H           Illegal character to local DCB
5EE4 115155      LD      DE,5551H        Address of local DCB
5EE7 12          LD      (DE),A          Store illegal character so OPEN
                                         will return immediately
5EE8 CD2444      CALL    4424H           OPEN dummy file
                                         (load SYS2 into 4E00)
```

Listing Continued . . .

. . . Continued Listing

```
5EEB C5          PUSH    BC          Save drive number
5EEC CDFD50      CALL    50FDH       See if drive available
                                     (use SYS2 subroutine)
5EEF 2067        JR      NZ,5F58H    Jump if drive not available
5EF1 79          LD      A,C         Fetch current drive number
5EF2 C630        ADD     A,30H       Convert drive number to ASCII
5EF4 32685F      LD      (5F68H),A   and move it to display line
```

Notice the dummy OPEN call at **5EE8**. This call is made simply to load SYS2, so the subroutine at **50FD** in SYS2 can be called. That subroutine determines if a drive is on line and ready, and returns a zero or non-zero status to indicate a ready or not-ready drive.

The code for step two is shown below:

Figure 9.35 *SYS6 Code From 5EF1 to 5F10*

```
5EF1 79          LD      A,C         Fetch current drive number
5EF2 C630        ADd     A,30H       Convert drive number to ASCII
5EF4 32685F      LD      (5F68H),A   and move it to display line
5EF7 CDF04A      CALL    4AF0H       Read GAT sector for drive
5EFA C24C55      JP      NZ,554CH    Jump if error GAT sector read
5EFD 21D04D      LD      HL,4DD0H    Addr of disk name in GAT sector
5F00 116E5F      LD      DE,5F6EH    Addr of disk name in display
5F03 010800      LD      BC,0008H    Number of bytes in disk name
5F06 EDB0        LDIR                Move name from GAT sector to display
5F08 11795F      LD      DE,5F79H    Addr of creation date on display
5F0B 010800      LD      BC,0008H    Number of bytes in date
5F0E EDB0        LDIR                Move date from GAT sector to display
5F10 21004D      LD      HL,4D00H    Start addr of GAT table in GAT sec
```

The code for step three begins at **5F16**. It begins by initializing the HL register to the beginning address of the GAT in the sector buffer and zeroing the DE register, which will serve as the available granule counter.

A loop control value is loaded into the B register at **5316**. Notice that this value is a fixed constant equal to the number of tracks per drive.

The main loop for counting available granules begins at **5F18** and extends to **5F23**. Each GAT entry (there is one per track — each entry contains two granule flags in Bits 0 and 1), is fetched, and Bits 0 and 1 are interrogated. When either is found to be zero (granule is available), the available granule count is incremented by the instruction at **5F1D**.

After all GAT entries have been processed, the available granule count is converted to ASCII and stored in the FREE header message. Following is the code used for step three.

Figure 9.36 *SYS6 Code From 5F10 to 5F2D*

```
5F10 21004D      LD      HL,4D00H     Start addr of GAT table in GAT sec
5F13 110000      LD      DE,0000H     Zero accumulator. Will hold
                                      number of free grans
5F16 0623        LD      B,23H        Max no. of GAT entries to examine
5F18 7E          LD      A,(HL)       Fetch a GAT entry
5F19 37          SCF                  Set CARRY so 1 is forced to bit 7
5F1A 1F          RRA                  Bit 0 (granule) to carry
5F1B 3801        JR      C,5F1EH      Go if gran 0, current track assigned
5F1D 13          INC     DE           If not, count 1 free granule
5F1E FEFF        CP      0FFH         Test if bit 1 (gran 1) available
5F20 20F7        JR      NZ,5F19H     Jump if granule 1 available
5F22 2C          INC     L            Bump to next track in GAT
5F23 10F3        DJNZ    5F18H        Loop till all tracks tested
*
*
*
5F25 EB          EX      DE,HL        Count of available grans to HL
5F26 11905F      LD      DE,5F90H     Array addr to save ASCII value
5F29 CD6C60      CALL    606CH        Convert count to ASCII and save
5F2C C1          POP     BC           Restore current drive no. to C
5F2D C5          PUSH    BC           and save for later use
```

The code for step four begins at **5F2E** and extends through **5F4F**. This code scans the HIT, counting the number of available slots. As with the loop in step three, the HL register contains the table address, while the counter is maintained in the DE register.

Step four begins by reading the HIT sector into a buffer at **4D00**. After initializing the HL and DE registers, the main loop (**5F3A - 5F49**) is entered.

Available slots in the HIT are indicated by a zero byte. Searching the HIT for the zero bytes is complicated by the fact that not all bytes in the HIT represent slots. Technically, the HIT contains 256 (dec.) bytes which could be used as slots; however, only 48 (dec.) of these are directory slot entries.

Slot entries are assigned in groups of 8, starting on the address boundaries: **XX40**, **XX60**, **XX80**,. . . **XXE0**. The bulk of the search loop is concerned with detecting when the end of a slot group has been reached and computing the address for the next group.

After a slot has been tested (**5F3A - 5F3E**) and the slot counter has been incremented (if appropriate), a test for the end of the slot group is performed. The code for this test occuppies **5F3F - 5F44**. A jump at **5F44** is taken if the end of the group has not been reached. If the end of the group has been reached, control falls through to the instructions **5F46 - 5F49**, where the next slot group address is computed, and simultaneously a test for the end of the HIT is performed.

When the end of the HIT is reached, the available slot count is converted to ASCII and stored in the FREE header message via the instructions at **5F4B - 5F4F**. Following is the code used for step four:

Figure 9.37 *SYS6 Code From 5F2E to 5F4F*

```
5F2E CD2E51      CALL    512EH          Read HIT sector (use SYS2 subroutine)
*
*
*       Prepare to Compute the Number of Files Assigned to
*             Diskette by Counting HIT Indices
*
*
5F31 C24C55      JP      NZ,554CH       Jump if error during read
5F34 21404D      LD      HL,4D40H       Start of HIT area for user files
5F37 110000      LD      DE,0000H       Clear accumulator
5F3A 7E          LD      A,(HL)         Fetch a HIT index
5F3B B7          OR      A              Set status flags
5F3C 2001        Jr      NZ,5F3FH       .   Go if file assigned this slot
5F3E 13          INC     DE             .   Else count 1 space available
5F3F 2C          INC     L              .   Bump to next slot in HIT
5F40 7d          LD      A,L            .   Prepare to test for
5F41 E6E7        AND     0E7H           .   end of slot range
5F43 BD          CP      L              .   Reached end of a slot range?
5F44 28F4        JR      Z,5F3AH        .   If not, go continue searching
5F46 C620        ADD     A,20H          .   Yes, bump to next slot area
5F48 6F          Ld      L,A            .   Form next HIT address in HL
5F49 20EF        JR      NZ,5F3AH       .   Loop if end of HIT not reached
5F4B EB          EX      DE,HL          Free HIT slots to HL for conversion
5F4C 11835F      LD      DE,5F83H       5 byte array addr for ASCII HIT slots
5F4F CD6C60      CALL    606CH          Convert free slots to ASCII decimal
```

Step five displays the header message and tests if all drives have been processed. Following is the code used for step five:

Figure 9.38 *SYS6 Code From 5F52 to 5F5F*

```
5F52 21625F      LD      HL,5F62H       Addr of display line DRIVE X-- ...
5F55 CD6744      CALL    4467H          Display title line with values
5F58 C1          POP     BC             Restore drive no. to C register
5F59 0C          INC     C              Bump to next drive
5F5A 79          LD      A,C            Move drive to A for
5F5B FE04        CP      04H            Compare. Test if all drives done
```

Listing Continued . . .

```
. . . Continued Listing
5F5D 208C       JR      nZ,5EEBH      Jump if more drives to process
5F5F C32D40     JP      402DH         Done, ret to SYS1 for next command
```

9.10 KILL Command

Processing for the KILL command is very straightforward. The file name to be KILL'ed is copied from the command line to a local DCB area using a SYS1 function; then the file is KILL'ed by a nucleus call to the KILL subroutine. Following is the code used for this command:

Figure 9.39 *SYS6 Code From 5914 to 5929*

```
*
*               KILL Command Processing
*
5914 115155     LD      DE,5551H      Local DCB address
5917 CD1C44     CALL    441CH         File name to local DCB (SYS1 routine)
591A C23A55     JP      nZ,553AH      Jump if illegal chars in file name
591D CD2444     CALL    4424H         OPEN file to KILL
5920 C24C55     JP      NZ,554CH      Jump if any error during OPEN
5923 CD2C44     CALL    442CH         KILL file
5926 C24C55     JP      NZ,554CH      Jump if error during KILL
5929 C32D40     JP      402DH         Reload SYS1 to wait for next command
```

For a description of the KILL subroutine see Section (to be inserted later)

9.11 LIB Command

The LIB command dislays the command list used by SYS1 to determine if a system command is being invoked. It does not list the commands embedded in SYS1, which are: BASIC2, DEBUG and TRACE.

Since SYS1 is loaded beneath SYS6, it can be directly accessed by SYS6 although this is not a recommended practice. The LIB routine is aware of the structure for the command list and skips over the SYS1 address for processing the commands. The following code is used for the LIB command:

Figure 9.40 *SYS6 Code From 533A to 5367*

```
*
*                       LIB Routine
*
533A 21BD4E     LD      HL,4EBDH      Addr of 2nd cmd list in SYS1
533D 0E04       LD      C,04H         .   No. of cmds printed per line
533F 3EC2       LD      A,0C2H        .   Compression code for 2 blanks
5341 CD3300     CALL    0033H         .   Display two blanks
5344 0606       LD      B,06H         .   No. chars in each cmd list entry
5346 7E         LD      A,(HL)        .   Get a char from command list
5347 23         INC     HL            .   Bump to next char in list
5348 cD3300     CALL    0033H         .   Display character
```

Listing Continued . . .

. . . Continued Listing

```
534B 10F9       DJNZ    5346H       .   Loop till all chars displayed
534D 3EC8       LD      A,0C8H      .   Compression code for 8 blanks
534F CD3300     CALL    0033H       .   Display 8 blanks
5352 23         INC     HL          .   Skip over addr of action routine
5353 23         INC     HL          .   in SYS1 for this command
5354 7E         LD      A,(HL)      .   Get 1st char of next command
5355 B7         OR      A           .   Get status for end of list test
5356 280A       JR      Z,5362H     .   Jump if end of list reached
5358 0D         DEC     C           .   Count no. of cmds on this line
5359 20E9       JR      NZ,5344H    .   Jump if line not full
535B 3E0D       LD      A,0DH       .   Else, skip to next line
535D CD3300     CALL    0033H       .   By displaying carriage return
5360 18DB       JR      533DH       .   Go top of loop, reset counts
5362 3E0D       LD      A,0DH       Done, skip
5364 CD3300     CALL    0033H       to next line
5367 C32D40     JP      402DH       and ret to SYS1 for next command
```

9.12 LIST Command

The LIST command dislays the contents of a file on the video. It appears from the code that the intent was to provide a command of the form:

Figure 9.41 *LIST Command Syntax*

```
LIST␣filespec␣(line = x' line #')
```

where line x is an optional parameter which specifies where the display is to begin. This syntax would work, except for a bug in the code where the command line is parsed for the LINE option. The error arises because the LINE text address is loaded into the BC register rather than the DE resister. The code reads:

Figure 9.42 *SYS6 Code From 5942 to 5945*

```
5942 018B59     LD      BC,598BH    Address of LINE text
5945 CD7644     CALL    4476H       SYS1, option 6 test for LINE option
```

whereas, it should be:

Figure 9.43 *SYS6 Code From 5942 to 5945*

```
5942            LD      DE, 598BH   Address of LINE text
5945            CALL    4476H       SYS1, option 6 test for line option
```

Assuming this error were corrected, LIST would work as follows:

The file name is copied to a local DCB using a SYS1 function. If a line number is specified, its value is stored in location **5954** - **5955**. The file is OPEN'ed, and the specified number of lines is skipped. The balance of the file is read on a character basis and displayed until an end of file or record not found status is returned. Following is the code used for the LIST command:

Figure 9.44 *SYS6 Code From 592C to 5988*

```
*
*               LIST Command Processing
*
592C 010000      LD      BC,0000H      Signal no LINE option
592F ED435459    LD      (5954H),BC    Selected by zeroing 5954
5933 115155      LD      DE,5551H      Address of local DCB
5936 CD1C44      CALL    441CH         Copy and edit file. Move it to
                                       local DCB. Use SYS1 subroutine.
5939 C23A55      JP      NZ,553AH      Jump if illegal chars in file name
593C 1A          LD      A,(DE)        Test for
593D FE2A        CP      2AH           Reference to special file (*)
593F CA3A55      JP      Z,553AH       Go if special file reference (error)
5948 0600        LD      B,00H         Specify physical I/O
594A 210056      LD      HL,5600H      Sector buffer address for file
594D CD2444      CALL    4424H         OPEN file to be LIST'ed
5950 C24C55      JP      NZ,554CH      JUMP if error during OPEN
5953 010000      LD      BC,0000H      LINE option flag
5956 78          LD      A,B           BC = 0000 if LINE not specified
5957 B1          OR      C             BC = Value if LINE value specified
5958 280D        JR      Z,5967H       Jump if LINE option not given
595A CD1300      CALL    0013H         Get a character from source file
595D FE0D        CP      0DH           Have we reached end of a line
595F 20F9        JR      NZ,595AH      No, loop till end of line
5961 0D          DEC     C             Yes, decrement line count
5962 20F6        JR      NZ,595AH      in BC. If non-zero
5964 05          DEC     B             Keep looping (skipping lines)
5965 20F3        JR      NZ,595AH      till no. in LINE option passed
5967 115155      LD      DE,5551H      Addr of file DCB. Begin listing file
596A CD1300      CALL    0013H         Get a character from source file
596D 200F        JR      NZ,597EH      Go if not good status (go test EOF)
596F CD3300      CALL    0033H         Display character returned
5972 FE0D        CP      0DH           Test for end of line
5974 20F1        JR      NZ,5967H      If not end, loop till end found
5976 CD1555      CALL    5515H         Test for BREAK/SHIFT@ condition
5979 C23040      JP      NZ,4030H      If BREAK active terminate LIST
597C 18E9        JR      5967H         Else, loop till end of source file
597E FE1C        CP      1CH           Test error status for EOF
5980 CA2D40      JP      Z,402DH       If so, Get SYS1 for next command
5983 FE1D        CP      1DH           Test for record not found error
5985 CA2D40      JP      Z,402DH       If RNF, done. Get SYS1
5988 C34C55      JP      554CH         Call SYS4 to generate error msg.
```

At **5976** a call is made to a subroutine at **5515**. This subroutine checks for an active SHIFT @ key, and if found, pauses until any other key is struck. This allows a user to suspend and resume a listing operation. Unfortunately, the function does not work reliably because the test for the SHIFT @ key is followed almost immediately by a test for any active key (resume). This can result in the SHIFT @ key being read twice, once for the suspend read and immediately afterwards for the resume read. The following code is used for the suspend/resume operation:

Figure 9.45 *SYS6 Code From 5515 to 5539*

```
5515 3A0F43      LD      A,(430FH)     Get system condition flags
5518 E620        AND     20H           Isolate BREAK key test flag
551A EE20        XOR     20H           Test if BREAK test flag on
551C C8          RET     Z             Ret to caller if BREAK test inactive
551D 3A4038      LD      A,(3840H)     Load row contents for BREAK key
5520 E604        AND     04H           Isolate BREAK key list
5522 C0          RET     NZ            Exit non-zero status if BREAK active
5523 CD2B00      CALL    002BH         BREAK not active. Strobe keyboard
5526 FE60        CP      60H           Test for SHIFT @
5528 2802        JR      Z,552CH       Jump if SHIFT @ active
552A AF          XOR     A             Else signal no activity
552B C9          RET                   and return to caller
552C 3A4038      LD      A,(3840H)     Refetch BREAK row contents
552F E604        AND     04H           Isolate BREAK key bit
5531 C0          RET     NZ            Exit if BREAK active
5532 CD2B00      CALL    002BH         If not active strobe keyboard again
5535 B7          OR      A             Set status flag for input
5536 28F4        JR      Z,552CH       Loop till BREAK active (return
                                       w/non-zero status)
5538 AF          XOR     A             Or, any key active (return
                                       w/zero status)
5539 C9          RET                   Return to caller
```

9.13 LOAD Command

Processing for the LOAD command begins at **5994**. The code begins by calling SYS1 with a option 40 to copy the file name from the LOAD command to a local DCB area. After the file name has been copied, the file loader in the nucleus is called to load the file. The following code is used for LOAD processing:

Figure 9.46 *SYS6 Code From 5994 to 59A9*

```
*
*                LOAD command processor
*
5994 115155      LD      DE,5551H      Address of local DCB area
5997 CD1C44      CALL    441CH         SYS1/code 4, copy file name to DCB
599A C23A55      JP      NZ,553AH      Jump if illegal chars found in name
599D 1A          LD      A,(DE)        else test for special file
```

Listing Continued . . .

. . . Continued Listing

```
599E FE2A       CP      2AH         name (*)
59A0 CA3A55     JP      Z,553AH     Go if special file name * (error)
59A3 CD3044     CALL    4430H       Call nucleus routine to load file
                                    (hope it doesn't overlay SYS6)
59A6 CA2D40     JP      Z,402DH     No error during load. Recall SYS1
59A9 C34C55     JP      554CH       Add file bit to error code and
                                    call SYS 4. Wait for next command
```

For details on the file loader in the nucleus see Section 4.3.8.

9.14 PRINT Command

The PRINT command is similar to the LIST command except that it has no LINE option, and the file is copied to the printer rather than the video. The code begins with a SYS1 call to move the file name from the command line to a local DCB area. Following that, the file is OPEN'ed and read on a character basis. Each character read is sent to the printer by calling the print driver in LEVEL II. Then a subroutine at **5515** is called in test for SHIFT @ or BREAK (see Section 9.12 for details).

The read, print, check for suspend loop terminates when an end of file, or record not found status is returned from the read operation. Following is the code used for the PRINT command:

Figure 9.47 *SYS6 Code From 59AC to 59EB*

```
*
*               PRINT Command Processor
*
59AC 115155     LD      DE,5551H    Address of local DCB area
59AF CD1C44     CALL    441CH       SYS1/code 4, copy file name to DCB
59B2 C23A55     JP      NZ,553AH    Go if illegal chars in name (error)
59B5 1A         LD      A,(DE)      Fetch 1st character of file name
59B6 FE2A       CP      2AH         Test for special file name (*)
59B8 CA3A55     JP      Z,553AH     Jump if special file name (error)
59BB 0600       LD      B,00H       Specify physical I/O
59BD 210056     LD      HL,5600H    Sector buffer address
59C0 CD2444     CALL    4424H       OPEN file
59C3 C24C55     JP      NZ,554CH    Jump if error occurred during OPEN
59C6 115155     LD      DE,5551H    Address of disk file DCB
59C9 CD1300     CALL    0013H       Get a character from source file
59CC 200C       JR      NZ,59DAH    If error while reading source file
                                    go test for EOF/RECORD NOT FOUND
59CE CD3B00     CALL    003BH       Send char to ROM print driver
59D1 CD1555     CALL    5515H       Test for BREAK or SHIFT @ condition
59D4 C23040     JP      NZ,4030H    If BREAK active exit print operation
                                    If SHIFT @ active wait for key hit
59D7 C3C659     JP      59C6H       Loop till disk read status non-zero
59DA F5         PUSH    AF          Save disk status
59DB 3E0D       LD      A,0DH       Terminate current line by
```

Listing Continued . . .

. . . Continued Listing

```
59DD CD3B00      CALL    003BH          Printing a carriage return
59E0 F1          POP     AF             Reload disk drive status
59E1 FE1C        CP      1CH            EOF on source file?
59E3 CA2D40      JP      Z,402DH        If so, ret to SYS1 (TRSDOS READY)
59E6 FE1D        CP      1DH            If not, check for RECORD NOT FOUND
59E8 CA2D40      JP      Z,402DH        If RNF ret to SYS1 for next command
59EB C34C55      JP      554CH          Not EOF/RNF, let SYS4 process it
```

9.15 PROT Command

This command alters the password for all non-system files on the specified drive. Besides a drive specification, it has three options. They are:

PW - Change diskette password.
UNLOCK- Remove password from all user files.
LOCK- Assign the diskette password to all files.

Code for PROT processing begins at **536A** and ends at **546B**. It begins by: testing a bit in **430F** (system condition flags) to determine if the PROT command is privileged (executable by the user), initializing the PW (**53D9**) LOCK (**53F9**) and UNLOCK (**5403**) bytes, processing a drive specification if present, and calling SYS1 (option 60) to parse the option in the PROT command line. The following code is used for these operations:

Figure 9.48 *SYS6 Code From 536A to 5395*

```
*
*                    PROT Command Processing
*
536A 3A0F43      LD      A,(430FH)      Get system condition/permission flag
536D CB6F        BIT     05H,A          Test if PROT command allowed
536F C26E54      JP      NZ,546EH       Jump if PROT not permitted
5372 010000      LD      BC,0000H       Zero value for
5375 ED43D953    LD      (53D9H),BC     PW flag
5379 ED43F953    LD      (53F9H),BC     LOCK flag
537D ED430354    LD      (5403H),BC     UNLOCK flag
5381 0E00        LD      C,00H          Setup C reg for default drive 0
5383 7E          LD      A,(HL)         Get 1st char following PROT command
5384 FE3A        CP      3AH            If it's a : there's a drive spec.
5386 2006        JR      NZ,538EH       Jump if no drive specification
5388 23          INC     HL             Get drive no. Bump to next char
5389 7E          LD      A,(HL)         Fetch drive number from command
538A D630        SUB     30H            Convert ASCII to binary
538C 4F          LD      C,A            Save drive number in C reg
538D 23          INC     HL             Bump to next char in command
538E 79          LD      A,C            Drive number to A register
538F 32BC53      LD      (53BCH),A      so it can be saved at 53BC
5392 11EB54      LD      DE,54EBH       Address of PW/LOCK/UNLOCK options
5395 CD7644      CALL    4476H          SYS1/code 6, compare rest of
                                        command to option list
```

Next: the result of the option parse is tested; SYS2 is loaded with a dummy OPEN call because its password encode subroutine will be needed, and the diskette password is read from the keyboard and compared to the password on the diskette. The following code is used for these operations:

Figure 9.49 *SYS6 Code From 5398 to 53D5*

```
5398 3AD953      LD      A,(53D9H)     Get PW flag
539B 21F953      LD      HL,53F9H      Address of LOCK flag
539E B6          OR      (HL)          Combine PW/LOCK test results (SYS1)
539F 210354      LD      HL,5403H      Address of UNLOCK flag
53A2 B6          OR      (HL)          Combine with PW/LOCK/UNLOCK test
                                       results from SYS1
53A3 CA6B52      JP      Z,526BH       Go if PW/LOCK/UNLOCK not specified
53A6 3E03        LD      A,03H         Illegal character code
53A8 115155      LD      DE,5551H      Address of local DCB
53AB 12          LD      (DE),A        Put illegal char in DCB so OPEN
                                       will return immediately
53AC CD2444      CALL    4424H         OPEN dummy file. Load SYS2 into 4E00
53AF 21A054      LD      HL,54A0H      Addr of MASTER PASSWORD? message
53B2 CD6744      CALL    4467H         Display message
53B5 CD5454      CALL    5454H         Go read password from keyboard
53B8 226851      LD      (5168H),HL    Save encode of password
53BB 0E00        LD      C,00H         C = drive number to read
53BD CDF04A      CALL    4AF0H         Read track 11, sector 0 into 4D00
53C0 C24C55      JP      NZ,554CH      Jump if error during READ
53C3 2ACE4D      LD      HL,(4DCEH)    Addr of password in GAT sector buffer
53C6 ED5B6851    LD      DE,(5168H)    Addr of new PW in SYS2 storage area
53CA AF          XOR     A             Clear CARRY flag
53CB ED52        SBC     HL,DE         Compare encode of passwords
53CD 2809        JR      Z,53D8H       Go if correct PW entered, 53CF 21D254
LD      HL,54D2H      Addr of INVALID MASTER PASSWORD msg
53D2 CD6744      CALL    4467H         Display message
53D5 C33040      JP      4030H         Return to SYS1 for next command
```

The diskette password is read by the following subroutine:

Figure 9.50 *SYS6 Code From 5454 to 546B*

```
5454 0608        LD      B,08H         Max no. of characters to read
5456 210056      LD      HL,5600H      Address of local buffer
5459 CD4000      CALL    0040H         Wait for user input
545C EB          EX      DE,HL         DE = beginning buffer address
545D 2600        LD      H,00H         Set HL equal to number of
545F 68          LD      L,B           characters read from keyboard
5460 19          ADD     HL,DE         Compute end of data in buffer
5461 3E08        LD      A,08H         Then compute number of bytes
5463 90          SUB     B             remaining in buffer
5464 47          LD      B,A           and save as counter
```

Listing Continued . . .

. . . Continued Listing

```
5465 3E20       LD      A,20H       Then blank fill
5467 77         LD      (HL),A      rest of buffer
5468 23         INC     HL          Bump to next position in buffer
5469 10FC       DJNZ    5467H       and loop till buffer blank
546B C3D150     JP      50D1H       SYS2 subroutine to encode password
```

Assuming no errors occurred in any of the code above, processing for each of the PROT options begin. The section of code for each option begins by testing if that particular option was specified. If it was not, control goes to the next option.

Following is the code used for the PW option:

Figure 9.51 *SYS6 Code From 53D8 to 53F2*

```
53D8 110000     LD      DE,0000H    Load results from SYS1 PW test
53DB 7A         LD      A,D         53D9/53DA will contain
53DC B3         OR      E           FFFF if PW option detected
53DD 2816       JR      Z,53F5H     Go if zero (PW option not selected)
53DF 21B954     LD      HL,54B9H    Addr of NEW MASTER PASSWORD? msg
53E2 CD6744     CALL    4467H       Display message
53E5 CD5454     CALL    5454H       Put reply (PW) in buffer at 5600
53E8 22CE4D     LD      (4DCEH),HL  Save PW encode in GAT sector buffer
53EB 3ABC53     LD      A,(53BCH)   Get drive number
53EE 4F         LD      C,A         Move to C register for nucleus call
53EF CD034B     CALL    4B03H       Write GAT sector
53F2 C24C55     JP      NZ,554CH    Go if error during GAT sector write
```

Following is the code for the LOCK and UNLOCK option. Both options cause the same operation to take place; the difference is that in one case the encode of the diskette password is added to all fields, and in the other case all passwords are removed (replaced by the encode of a password for eight blanks).

Figure 9.52 *SYS6 Code From 53F5 to 5451*

```
53F5 2ACE4D     LD      HL,(4DCEH)  Load encode of password
53F8 110000     LD      DE,0000H    Get SYS flag for LOCK option
53FB 7A         LD      A,D         53F9-53FA will contain
53FC B3         OR      E           FFFF if LOCK specified
53FD 200D       JR      NZ,540CH    Jump if LOCK specified
53FF 219642     LD      HL,4296H    Encode of UNIVERSAL password
5402 110000     LD      DE,0000H    Get SYS1 flag for UNLOCK
5405 7A         LD      A,D         Combined SYS1
5406 B3         OR      E           flag bytes
5407 2003       JR      NZ,540CH    Go if UNLOCK specified, else
5409 C32D40     JP      402DH       Go SYS1 (TRSDOS READY) No PROT cmd
540C 223054     LD      (5430H),HL  Save LOCK/UNLOCK PW encode specified
540F 3ABC53     LD      A,(53BCH)   Load drive number
```

Listing Continued . . .

. . . Continued Listing

```
5412 4F          LD      C,A          Move to C for nucleus call
5413 CD554B      CALL    4B55H        Get dir track no. for drive in C
5416 1E02        LD      E,02H        Sector number to read
5418 210042      LD      HL,4200H     Buffer address = 4200
541B CD354B      CALL    4B35H        Read track 11, sector 2
541E C24C55      JP      NZ,554CH     Jump if error during read
5421 2E40        LD      L,40H        Form addr of 3rd entry in directory
5423 7E          LD      A,(HL)       Get entry available/assigned byte
5424 E6F8        AND     0F8H         Isolate available/assigned list
5426 FE10        CP      10H          Test if entry available
5428 2010        JR      NZ,543AH     Jump if entry assigned
542A 7D          LD      A,L          Get LSB of beginning of entry addr
542B C610        ADD     A,10H        Compute addr of UPDATE password
542D 6F          LD      L,A          Reform address in HL
542E D5          PUSH    DE           Save track/sector number
542F 110000      LD      DE,0000H     Load master password
5432 73          LD      (HL),E       Move it to UPDATE password (LSB)
5433 2C          INC     L            Bump to next password in entry
5434 72          LD      (HL),D       MSB of master pw to MSB of UPDATE pw
5435 2C          INC     L            Bump to LSB of access pw
5436 73          LD      (HL),E       LSB of master pw to LSB of access pw
5437 2C          INC     L            Bump to MSB of PW
5438 72          LD      (HL),D       MSB of master PW to MSB of ACCESS PW
5439 D1          POP     DE           Restore track/sector no. for next read
543A 7D          LD      A,L          Prepare to form addr of next entry
543B E6E0        AND     0E0H         Test if end of sector reached
543D C620        ADD     A,20H        Compute address of next entry
543F 6F          LD      L,A          Form address in HL
5440 30E1        JR      NC,5423H     Go if end of buffer not reached
5442 210042      LD      HL,4200H     Reform buffer address
5445 CDEF46      CALL    46EFH        Write updated sector back to disk
5448 C24C55      JP      NZ,554CH     Jump if error during rewrite
544B 1C          INC     E            Bump sector address
544C 7B          LD      A,E          and move it to A where end
544D FE0A        CP      0AH          of track test can be made
544F 38C7        JR      C,5418H      Jump if not end of track
5451 C32D40      JP      402DH        Done, ret to SYS1 for next command
```

9.16 RENAME Command

This command changes the name of the first file specified to the second file name specified. Processing for this command begins at **5F9C**.

Processing begins with a call to SYS1 to copy the first file name (the one to be changed) to a local DCB (**5551**). Next, the file named as the first parameter is OPEN'ed, and the second file name is copied to a local DCB (**5571**) using a SYS1 call. The following code is used for these operations:

Figure 9.53 *SYS6 Code From 5F9C to 5FC3*

```
*
*                    RENAME Command Processing
*
5F9C 115155      LD      DE,5551H      Load DCB address
5F9F CD1C44      CALL    441CH         SYS1/code 4, copy file name to DCB
5FA2 C23A55      JP      NZ,553AH      Jump if illegal chars in name
5FA5 1A          LD      A,(DE)        Fetch 1st character of name
5FA6 FE2A        CP      2AH           Test for special system file (*)
5FA8 CA3A55      JP      Z,553AH       Error if found (file spec. required)
5FAB CD2444      CALL    4424H         OPEN file, load SYS2 at 4E00
5FAE C24C55      JP      NZ,554CH      Jump if file not present
5FB1 117155      LD      DE,5571H      Local DCB addr for 2nd file name
5FB4 CD1C44      CALL    441CH         SYS1/code 4, Copy file name to DCB
5FB7 C23A55      JP      NZ,553AH      Jump if illegal chars in new name
5FBA 3A5255      LD      A,(5552H)     Get access flags for file to RENAME
5FBD E607        AND     07H           Isolate access permission
5FBF FE03        CP      03H           Test for RENAME permission
5FC1 3E25        LD      A,25H         Error code for not allowed
5FC3 D24C55      JP      NC,554CH      Go if RENAME permission required
```

Next, the second file is tested for a drive specification. Assuming none exists, the drive number from the OPEN'ed DCB for the first file is appended to the second file name. The second file is then OPEN'ed, and a check is made to insure the new file name is not already assigned. The following code is used for these operations:

Figure 9.54 *SYS6 Code From 5FC6 to 5FF3*

```
*
*       Scan file name looking for a drive specification.
*           If one is found an error has occurred.
*
5FC6 217155      LD      HL,5571H      .   Addr of DCB with new file name
5FC9 7E          LD      A,(HL)        .   Get a character from file name
5FCA FE3A        CP      3AH           .   Test for : (drive spec)
5FCC 285A        JR      Z,6028H       .   Go if : found (error)
5FCE FE20        CP      20H           .   Test for end of name
5FD0 3803        JR      C,5FD5H       .   Jump if end of file name
5FD2 23          INC     HL            .   Else bump to next character
5FD3 18F4        JR      5FC9H         .   and loop till end of name
5FD5 363A        LD      (HL),3AH      Then add drive specification
5FD7 23          INC     HL            Bump to next position in DCB
5FD8 3A5755      LD      A,(5557H)     Fetch drive number (in binary)
```

Listing Continued . . .

. . . Continued Listing

```
5FDB C630       ADD     A,30H       Convert to ASCII
5FDD 77         LD      (HL),A      and save in DCB after new name
5FDE 23         INC     HL          Bump to first pos. past drive spec.
5FDF 3603       LD      (HL),03H    and store a terminator
5FE1 117155     LD      DE,5571H    Addr of DCB with new file name
5FE4 CD2444     CALL    4424H       OPEN new file name
5FE7 CA4E60     JP      Z,604EH     Go if name already exists (error)
5FEA FE18       CP      18H         Test status for other error
5FEC C24C55     JP      NZ,554CH    Jump if some other error occurred
5FEF ED4B5755   LD      BC,(5557H)  Drive no. to C, dir sector # to B
5FF3 C5         PUSH    BC          Save drive number
```

After the operations above have been completed without error, the directory sector for the file to be renamed is read using a nucleus subroutine. The new file name is copied over the previous file name in the sector buffer and the sector is rewritten. The following code is used for these operations:

Figure 9.55 *SYS6 Code From 5FEF to 600B*

```
5FEF ED4B5755   LD      BC,(5557H)  Drive no. to C, dir sector # to B
5FF3 C5         PUSH    BC          Save drive number
5FF4 CDC14A     CALL    4AC1H       Read dir sector for file to rename
5FF7 C24C55     JP      NZ,554CH    Go if error during dir read
5FFA 54         LD      D,H         Form addr for file entry in DE
5FFB 7D         LD      A,L         by
5FFC C605       ADD     A,05H       adding 5 to the address in HL
5FFE 5F         LD      E,A         and moving it to DE
5FFF 215D51     LD      HL,515DH    Addr of SYS2 DCB with new file name
6002 010B00     LD      BC,000BH    Max no. of chars in new file name
6005 EDB0       LDIR                Copy new name from buffer in SYS2
                                    to directory file name area
6007 C1         POP     BC           Restore dir sector/drive no.
6008 CDD64A     CALL    4AD6H       Rewrite directory sector
600B C24C55     JP      NZ,554CH    Jump if error during write
```

After the directory sector has been updated, the HIT sector is read, and the hash code for the new file name is stored at the HIT index for the original file name. Following that, the HIT sector is rewritten, and control is returned to SYS1 to wait for the next command. The following code is used for these operations:

Figure 9.56 *SYS6 Code From 600E to 6025*

```
600E CD2E51     CALL    512EH       Use SYS2 subroutine to read HIT
6011 C24C55     JP      NZ,554CH    Jump if error while reading HIT
6014 54         LD      D,H         H = MSB of HIT sector buffer.
                                    B = directory sector number
6015 58         LD      E,B         DE = addr of HIT slot for this file
```

Listing Continued . . .

. . . Continued Listing

```
6016 C5          PUSH    BC              Save directory sector/drive number
6017 215D51      LD      HL,515DH        Addr of SYS2 buffer with new name
601A CD9B50      CALL    509BH           Compute hash code for new file.
601D C1          POP     BC              Restore dir sector/drive number
601E 12          LD      (DE),A          Save new hash code
601F CD4151      CALL    5141H           SYS2 subroutine to write updated HIT
6022 C24C55      JP      NZ,554CH        Jump if error while writing HIT
6025 C32D40      JP      402DH           Done, load SYS1 for next command
```

9.17 TIME Command

This command reads the time following the TIME command and copies it to the time buffer in the nucleus area (**4041**). A subroutine at **52EA** is called to process the time declaration. The operation of this subroutine is discussed in Section 9.6.

Following is the code used for TIME command processing:

Figure 9.57 *SYS6 Code From 52B1 to 52C4*

```
*
*                       TIME Routine
*
52B1 0E3A        LD      C,3AH           C = : (only valid terminator)
52B3 CDEA52      CALL    52EAH           Process time. Leave in 52DC-52DE
52B6 C2D352      JP      NZ,52D3H        Go if anything but number or :
52B9 21DC52      LD      HL,52DCH        Addr of local buffer containing time
52BC 114140      LD      DE,4041H        Addr of time buffer in system area
52BF 010300      LD      BC,0003H        Number of bytes to move
52C2 EDB0        LDIR                    Move time to system area
52C4 C9          RET                     Ret to SYS6 to read next command
```

9.18 VERIFY Command

This command enables or disables a verify (read) after all write operations. It begins by calling a subroutine in SYS1 to parse the ON/OFF option of the command, and depending on the option present, modifies the address of the WRITE vector in the nucleus to jump to either the WRITE subroutines or the VERIFY subroutines. The folowing code is used for this command.

Figure 9.58 *SYS6 Code From 5504 to 5512 and 60CD to 60D8*

```
*
*              Verify Command Processing
*
5504 CDA051      CALL    51A0H           SYS1 routine to check for ON/OFF
5507 218B47      LD      HL,478BH        Turn verify flag ON or OFF
550A 2803        JR      Z,550FH         Jump if VERIFY (OFF)
550C 21A847      LD      HL,47A8H        Move addr of VRFY routine in nucleus
```

Listing Continued . . .

. . . Continued Listing

```
550F 223A44      LD      (443AH),HL   to addr field of WRITE jump vector
5512 C32D40      JP      402DH         Done. Ret to SYS1 for next command
*
*
*
60CD 218B47      LD      HL,478BH     Address of WRITE routine in nucleus
60D0 2803        JR      Z,60D5H      Go clear verify flag if (OFF) option
60D2 AF          XOR     A            Zero A, clear status flags
60D3 3C          INC     A            Load 1 into A (auto verify writes)
60D4 B7          OR      A            Set status flags non-zero
60D5 325E48      LD      (485EH),A    Save verify flag in nucleus
60D8 C9          RET                  Return to caller
```

notes

10

10.0 BOOT/SYS

BOOT/SYS is the program used to load the nucleus routine SYS0/SYS. It is loaded and executed by LEVEL II if disks are detected during RESET processing.

BOOT/SYS is an absolute core image program residing on sector 0 of track 0. Since it is a core image program (in fact it is the only one in TRSDOS), BOOT/SYS can be read directly into memory and executed. Following is the LEVEL II code used to load BOOT/SYS

Figure 10.1 *LEVEL II BASIC Code From 0 to 2, 66 to 72 and 674 to 6C9*

```
0000 F3       DI                --- Power on IPL entry -Turn off clock/disk interrupts
0001 AF       XOR     A         --- Clear A-reg, status
0002 C37406   JP      0674H     --- Go to beginning of IPL sequence


0066 310006   LD      SP,0600H  --- Reset IPL entry
0069 3AEC37   LD      A,(37ECH) --- Get controller status
006C 3C       INC     A         --- Test for controller present
006D FE02     CP      02H       --- Status usually FF if no EI
006F D20000   JP      NC,0000H  --- NC if controller addressable. Join common IPL
0072 C3CC06   JP      06CCH     --- No disk go to BASIC 'READY' prompt


0674 D3FF     OUT     (0FFH),A  --- 0 to cassette ************** Video controller ****
0676 21D206   LD      HL,06D2H  --- Addr. of video/keyboard/printer DCB's
0679 110040   LD      DE,4000H  --- Start of communications region
067C 013600   LD      BC,0036H  --- Setup for block move
067F EDB0     LDIR              --- Move 6D2-707 to 4000-4035
0681 3D       DEC     A         --- Change value being sent to port FF to FFFD, . . .
```

Listing Continued . . .

. . . Continued Listing

```
0682 3D       DEC     A           --- FFFB, . . . .
0683 20F1     JR      NZ,0676H    --- Go thru this 128 times
0685 0627     LD      B,27H       --- 0 to A
0687 12       LD      (DE),A      --- 0 to 4036-4062
0688 13       INC     DE          --- Bump destination pntr
0689 10FC     DJNZ    0687H       --- Go if not done
068B 3A4038   LD      A,(3840H)   --- Test keyboard for BREAK
068E E604     AND     04H         --- BREAK key hit
0690 C27500   JP      NZ,0075H    --- Go if BREAK
0693 317D40   LD      SP,407DH    --- New stack area
0696 3AEC37   LD      A,(37ECH)   --- Load disk status
0699 3C       INC     A           --- Test for Expansion Interface
069A FE02     CP      02H         --- and disk drive
069C DA7500   JP      C,0075H     --- Go if no disk
069F 3E01     LD      A,01H       --- Unit select mask for drive 0
06A1 32E137   LD      (37E1H),A   --- Select drive 0
06A4 21EC37   LD      HL,37ECH    --- Addr of disk command / status register
06A7 11EF37   LD      DE,37EFH    --- Addr of disk data register
06AA 3603     LD      (HL),03H    --- 3 to disk command register = restore, position
06AC 010000   LD      BC,0000H    --- Delay count                          :to track 0
06AF CD6000   CALL    0060H       --- Delay for approx 3 seconds
06B2 CB46     BIT     00H,(HL)    --- Test if controller busy,
06B4 20FC     JR      NZ,06B2H    --- Loop till not busy
06B6 AF       XOR     A           --- 0 to A
06B7 32EE37   LD      (37EEH),A   --- 0 to sector register
06BA 010042   LD      BC,4200H    --- BC = addr of buffer area
06BD 3E8C     LD      A,8CH       --- A = read command
06BF 77       LD      (HL),A      --- Read sector 0, track 0 into 4200 - 4455
06C0 CB4E     BIT     01H,(HL)    --- Test if data ready
06C2 28FC     JR      Z,06C0H     --- Go if no data avail
06C4 1A       LD      A,(DE)      --- Get next byte from disk
06C5 02       LD      (BC),A      --- Transfer data to 4200+
06C6 0C       INC     C           --- Bump buffer pntr
06C7 20F7     JR      NZ,06C0H    --- Go if not 256 bytes
06C9 C30042   JP      4200H       --- Done, transfer to TRSDOS loader
```

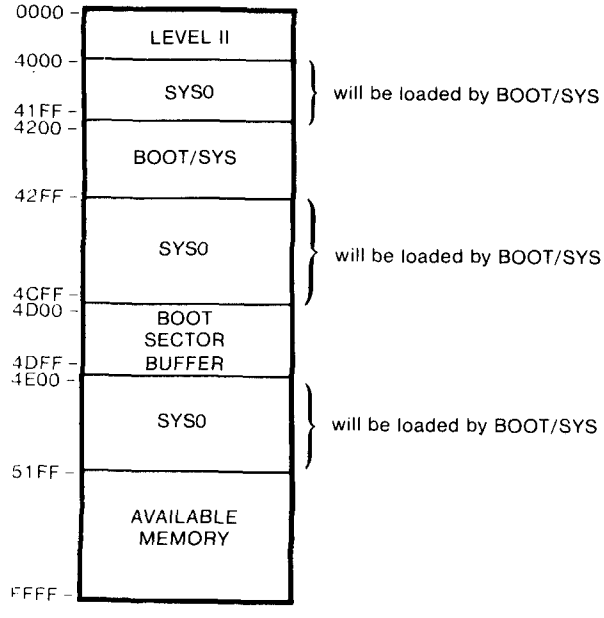
Immediately after the load operation completes, LEVEL II passes control to BOOT/SYS, which in turn will load SYS0/SYS. This process is more complex than the loading of BOOT/SYS for several reasons. First, the disk address for SYS0 is not fixed. Therefore, the directory entry for SYS0 must be read in order to determine its disk residency address. Second, SYS0/SYS contains loader control codes which must be intrepreted by BOOT/SYS. This means SYS0 cannot simply be read into memory. It must be read sector by sector into a buffer so the control codes can be separated from the binary image of the program.

The code for BOOT/SYS occupies locations **4200 - 42FF** while its buffer resides at **4D00 - 4DFF**. Both of these address ranges are within SYS0; however, those regions are used by SYS0 as sector buffers, so there will be no conflict during the load operation. Nonetheless, this practice is unusual, and it could present problems if either BOOT/SYS were extended or the sector buffers were used to carry constants or code to be used during the initialization process.

Execution of BOOT/SYS begins at **4200**. The instructions from **4200 - 4202** are confusing and serve no purpose. Byte 2 contains the directory track number. Other parts of the system may read the boot loader just for the directory track number. Byte 0 is non-zero during BACKUP operations. In any case, these instructions are benevolent and do not effect the remainder of the code.

Memory usage by BOOT/SYS is shown below:

Figure 10.2 *BOOT/SYS Memory Residency*



10.1 BOOT/SYS Execution

Execution of BOOT/SYS begins at **4200**. The instructions from **4200 - 4202** are confusing and serve no purpose. This confusion was either a deliberate effort to thwart intrepretation of the code or the design of the boot loader was changed at some point, and those instructions were never removed. In any case, they are benevolent and do not effect the remainder of the code.

Meaningful processing begins at **4203** with a Disable Interrupt instruction followed by code to initialize the stack pointer and clear the screen. Interrupts must be inhibited because Programmed I/O (PIO) instructions are used to read the disk during the load operation. Interrupts could disrupt the synchronization between the PIO instructions and the disk controller, causing data to be lost.

Next, drive zero is selected, and the registers are initialized for a call to a subroutine at **42AA** to read the directory sector containing the entry for SYS0. Note, this code assumes the directory entry for SYS0 will occupy sector four of the directory track.

After control returns from the read subroutine, a test for disk errors is made. If any occurred, the message 'DISK ERROR' is displayed and a HALT instruction is executed.

If no errors were detected, the first byte of the directory sector is tested to determine if an active entry is present. If not, the message 'NO SYSTEM' is displayed and a HALT instruction is executed.

Otherwise, the entry is assumed to belong to SYS0. At this point, the alternate register set is initialized with the beginning track number and sector offset for SYS0. Arithmetic to translate the sector offset into the starting sector number for the file is also performed. In addition, the alternate BC register pair is initialized to indicate an empty sector buffer. Following these operations, the main loop of the boot loader is entered. The code used for these operations is shown below.

Figure 10.3 *BOOT/SYS Code From 4200 to 423D*

```
4200 00          NOP                 BOOT/SYS flag. Used by BACKUP
4201 FE11        CP       11H        Directory track number
4203 F3          DI                  Disk I/O timing must
                                     not be interrupted
4204 31FC41      LD       SP,41FCH   Initialize stack pointer address
4207 21E242      LD       HL,42E2H   Address of "CLS" video routine
420A CD9A42      CALL     429AH      Issue video commands
420D 3E01        LD       A,01H      Unit select mask for drive 0
420F 32E137      LD       (37E1H),A  Select drive 0
4212 3A0242      LD       A,(4202H)  Fetch directory track number
4215 57          LD       D,A        Put it in the D register
4216 1E04        LD       E,04H      Put SYS0 directory no. in E
4218 01004D      LD       BC,4D00H   Local buffer address
421B CDAA42      CALL     42AAH      Get SYS0 dir entry (trk 11/sec 4)
421E 2070        JR       NZ,4290H   Jump if any error during READ
4220 3A004D      LD       A,(4D00H)  Get 1st byte of SYS0 dir entry
4223 E610        AND      10H        Isolate entry occupied flag
4225 21E542      LD       HL,42E5H   Address of DISK ERROR message
4228 2869        JR       Z,4293H    If no entry then go display NO SYSTEM
*
*   Initialize alternate DE and BC registers for get next
*   byte routine from sector buffer at 4275. DE contains
*   the track and sector number for the next sector to be
*   read. BC contains the current buffer address. This code
*   initializes BC to 4DFF which signals an empty sector
*   buffer.
*
422A D9          EXX                 Get alternate register set
422B 2A164D      LD       HL,(4D16H) Fetch 1st GAP for SYS0/SYS
422E 55          LD       D,L        D = track number for SYS0
422F 7C          LD       A,H        A = byte count, sector offset
4230 07          RLCA                Isolate bit 5 (sector offset)
4231 07          RLCA                and position it
4232 07          RLCA                over bit 0 so we can
4233 E607        AND      07H        isolate bit 5 (bits 6/7 are zero)
4235 67          LD       H,A        Put in H to multiply micro-style
4236 07          RLCA                Sector offset bit * 2
4237 07          RLCA                times 4
```

Listing Continued . . .

. . . Continued Listing

```
4238 84          ADD     A,H          times 5 gives 1st sector no. (Ø/5)
4239 5F          LD      E,A          Starting sector no. to E
423A Ø1FFAD      LD      BC,4DFFH     Signal empty buffer to force read
                                      on next buffer access
423D D9          EXX                  Switch back to main register set
```

The main loop for loading the nucleus is quite simple. The binary file of SYS0 is read on a byte-by-byte basis. All control codes except **01** and **02** are discarded.

When an **01** control code is encountered, the next 'n' bytes of data are read and stored at the address embedded in the control stream. Both the 'n' byte count and the load address for the file follow the **01** control code.

An **02** control code is treated as an end of file. When this code is encountered, control is passed to the address following the control code. In the case of SYS0, this address will be **4E00**, which is the beginning of the system initialization for TRSDOS.

The main loop for loading SYS0 begins at **423E**. It relies on a subroutine at **4275** to return the next byte from the sector buffer for the SYS0 file. The initial call to this subroutine is made to get a control byte. Subsequent calls are made to fetch supporting control bytes and data bytes.

Depending on the initial byte fetched, control either falls into the code at **4244** or branches to **425B**. After the processing for the particular control byte has completed, the main loop is re-entered at **423E**. Following is the code used for the main loop.

Figure 10.4 *BOOT/SYS Code From 423E to 4274*

```
*
*                ----  Main Loop  ----
*
423E CD7542      CALL    4275H        Get next byte from sector buffer.
                                      Must be a loader code
4241 3D          DEC     A            Is it code 1? (load data follows)
4242 2Ø17        JR      NZ,425BH     Jump if not control code 1
4244 CD7542      CALL    4275H        else get no. of bytes to load
4247 47          LD      B,A          Put load count in B for DJNZ instr.
4248 CD7542      CALL    4275H        Get LSB of load address
424B 6F          LD      L,A          Form load address in HL
424C Ø5          DEC     B            Count LSB as a byte loaded
424D CD7542      CALL    4275H        Get MSB of load address and
4250 67          LD      H,A          Form load address in HL
4251 Ø5          DEC     B            Count MSB as a byte loaded
4252 28EA        JR      Z,423EH      Go if no more bytes to load
4254 CD7542      CALL    4275H        Get a byte of load data
4257 77          LD      (HL),A       Save it at current load address
4258 23          INC     HL           Bump load address and
4259 18F6        JR      4251H        loop till byte count exhausted
```

Listing Continued . . .

. . . Continued Listing

```
425B 3D          DEC     A              Test for code 2 (transfer addr flag)
425C 28ØB        JR      Z,4269H        Jump if control code 2
425E CD7542      CALL    4275H          Not code 1 or 2 so ignore it and
4261 47          LD      B,A            get the byte count (bytes to discard)
4262 CD7542      CALL    4275H          Fetch a byte to discard
4265 1ØFB        DJNZ    4262H          Go get another till all discarded
4267 18D5        JR      423EH          Go get next control code
4269 CD7542      CALL    4275H          Code 2. Skip over byte count
426C CD7542      CALL    4275H          Fetch LSB of transfer address
426F 6F          LD      L,A            Move it to L
4270 CD7542      CALL    4275H          Fetch MSB of transfer address
4273 67          LD      H,A            Form transfer addresss in HL and
4274 E9          JP      (HL)           jump (entry point for SYSØ/SYS
```

The subroutine at **4275** fetches the next data byte from the sector buffer and returns it to the caller. The alternate register set is used by this subroutine; however, the data byte is returned in the A register of the primary register set.

A pointer into the sector buffer is maintainted in the BC register pair. Before a byte is fetched from the buffer, the C register is incremented to the next position in the buffer. If the buffer is empty (ZERO status), control falls into the code at **4279**, where the next sector from the file is read.

If the buffer is not empty, the next byte is fetched, and control returns to the caller.

When the buffer is empty, the next sector is read from the file through a call to a subroutine at **42AA**. Upon return from this call, a test for disk errors is made. If any errors occurred, control passes to **4290**, where the message 'DISK ERROR' is displayed and a HALT instruction is executed.

If there were no disk errors, the sector number is incremented and compared to a 10. This is a test for track overflow. If the sector number has reached 10, it is reset to zero, and the track number is incremented. Note, this code assumes SYS0 occuppies consecutive tracks, and is recorded in signle-density (10 sectors per track vs. 18 sectors per track). After the sector and track numbers have been updated, the pointer to the sector buffer is reset to zero (full buffer), and the first byte from the buffer is loaded before returning to the caller.

Following is the code used for the subroutines at **4275** and **42AA**:

Figure 10.5 *BOOT/SYS Code From 4275 to 422F and From 42AA to 42E1*

```
*
*          ----  Gets Next Byte From Sector Buffer  ----
*
4275 D9          EXX                    Switch to alternate register set
4276 ØC          INC     C              Bump buffer addr to next byte
4277 2Ø14        JR      NZ,428DH       Jump if sector buffer not empty
4279 C5          PUSH    BC             Save buf addr, prepare to read sector
427A 3EØ1        LD      A,ØlH          Unit select mask for drive Ø
```

Listing Continued . . .

. . . Continued Listing

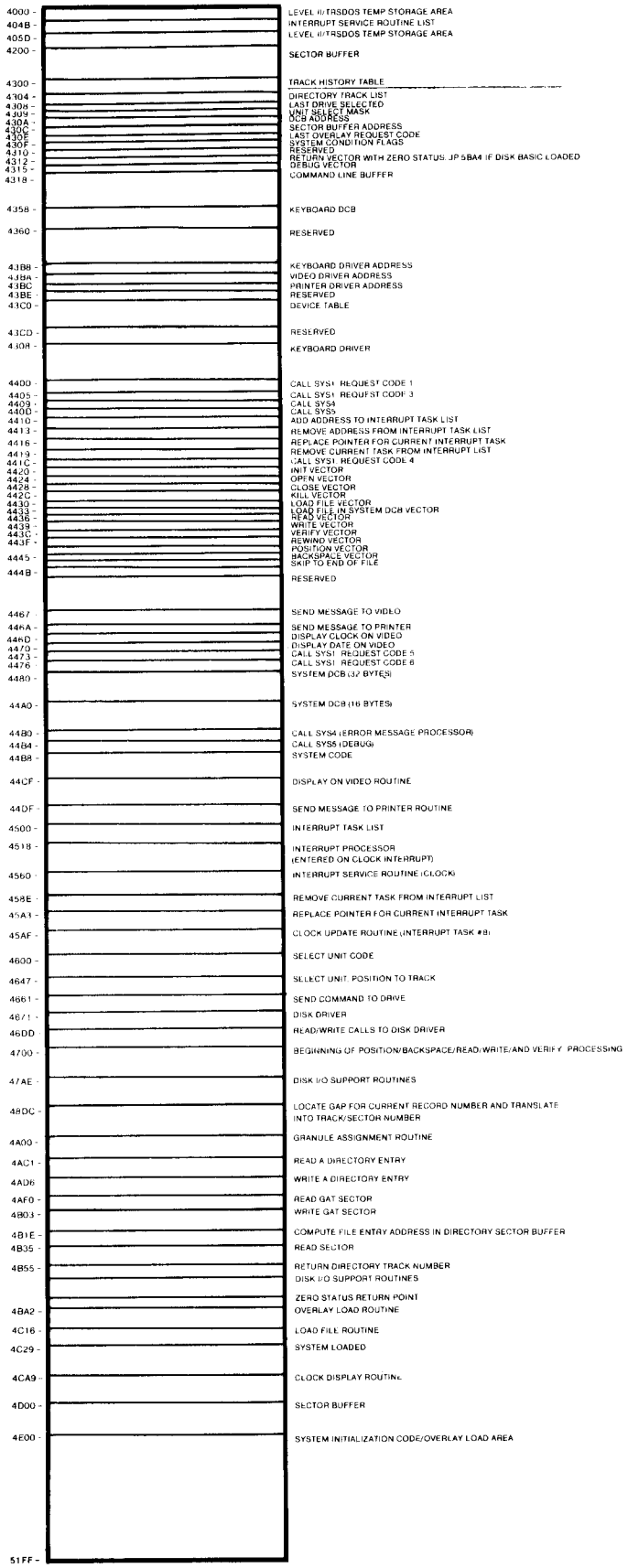
```
427C 32E137      LD      (37E1H),A     Re-select Ø to renew MOTOR ON cycle
427F CDAA42      CALL    42AAH         Read 1 sector (trk/sec no. in DE)
4282 200C        JR      NZ,4290H      Jump if any error during read
4284 C1          POP     BC            Restore sector buffer address
4285 1C          INC     E             Bump sector number, then
4286 7B          LD      A,E           test to see if the last sector
4287 D60A        SUB     0AH           of the current track has been read
4289 2002        JR      NZ,428DH      Jump if it hasn't
428B 5F          LD      E,A           else reset sector no. to zero
428C 14          INC     D             and bump current track number
428D 0A          LD      A,(BC)        Get next byte from buffer (last byte
                                       if 428D not jumped to by 4277)
428E D9          EXX                   Switch back to main register set
428F C9          RET                   and return to caller


42AA C5          PUSH    BC            Save caller's BC (sec buf addr)
42AB CDB242      CALL    42B2H         Read sector specified in DE
42AE E1          POP     HL            Caller's BC reg (sec buf addr)
42AF C8          RET     Z             Return if no disk error
42B0 44          LD      B,H           Disk error, retry operation
42B1 4D          LD      C,L           after restoring caller's BC
42B2 ED53EE37    LD      (37EEH),DE    Load track/sector regs in controller
42B6 21EC37      LD      HL,37ECH      Addr of stat/cmd registers for disk
42B9 361B        LD      (HL),1BH      Issue SEEK command
42BB F5          PUSH    AF            Delay so controller
42BC F1          POP     AF            has time to respond
42BD F5          PUSH    AF            to the command
42BE F1          POP     AF            before we ask for the status
42BF 7E          LD      A,(HL)        Fetch controller status
42C0 0F          RRCA                  Controller busy flag to carry flag
42C1 38FC        JR      C,42BFH       Loop till controller not busy
42C3 3688        LD      (HL),88H      Give controller READ SECTOR command
42C5 D5          PUSH    DE            Save track/sector number
42C6 11EF37      LD      DE,37EFH      Addr of controller's data register
42C9 C5          PUSH    BC            Delay for controller, so
42CA C1          POP     BC            It can react to last command
42CB 180B        JR      42D8H         Simulate CALL 42D0 to get data
                                       byte (or OP done status bit)
42CD 0F          RRCA                  Controller busy status to carry
42CE 300A        JR      NC,42DAH      Go if OP done (not busy status)
42D0 7E          LD      A,(HL)        Get status
42D1 CB4F        BIT     01H,A         Test DATA READY status bit
42D3 28F8        JR      Z,42CDH       Jump if data not available
42D5 1A          LD      A,(DE)        Read a data byte from controller
42D6 02          LD      (BC),A        Save in sector buffer
42D7 03          INC     BC            Bump sector buffer address
42D8 18F6        JR      42D0H         Loop till controller not busy
42DA 7E          LD      A,(HL)        Fetch controller status
42DB E65C        AND     5CH           Isolate error flags
42DD D1          POP     DE            Restore track/sector address
42DE C8          RET     Z             Ret if no error. If error occurred,
42DF 36D0        LD      (HL),0D0H     stop controller with force interrupt
42E1 C9          RET                   and then return to caller
```

notes

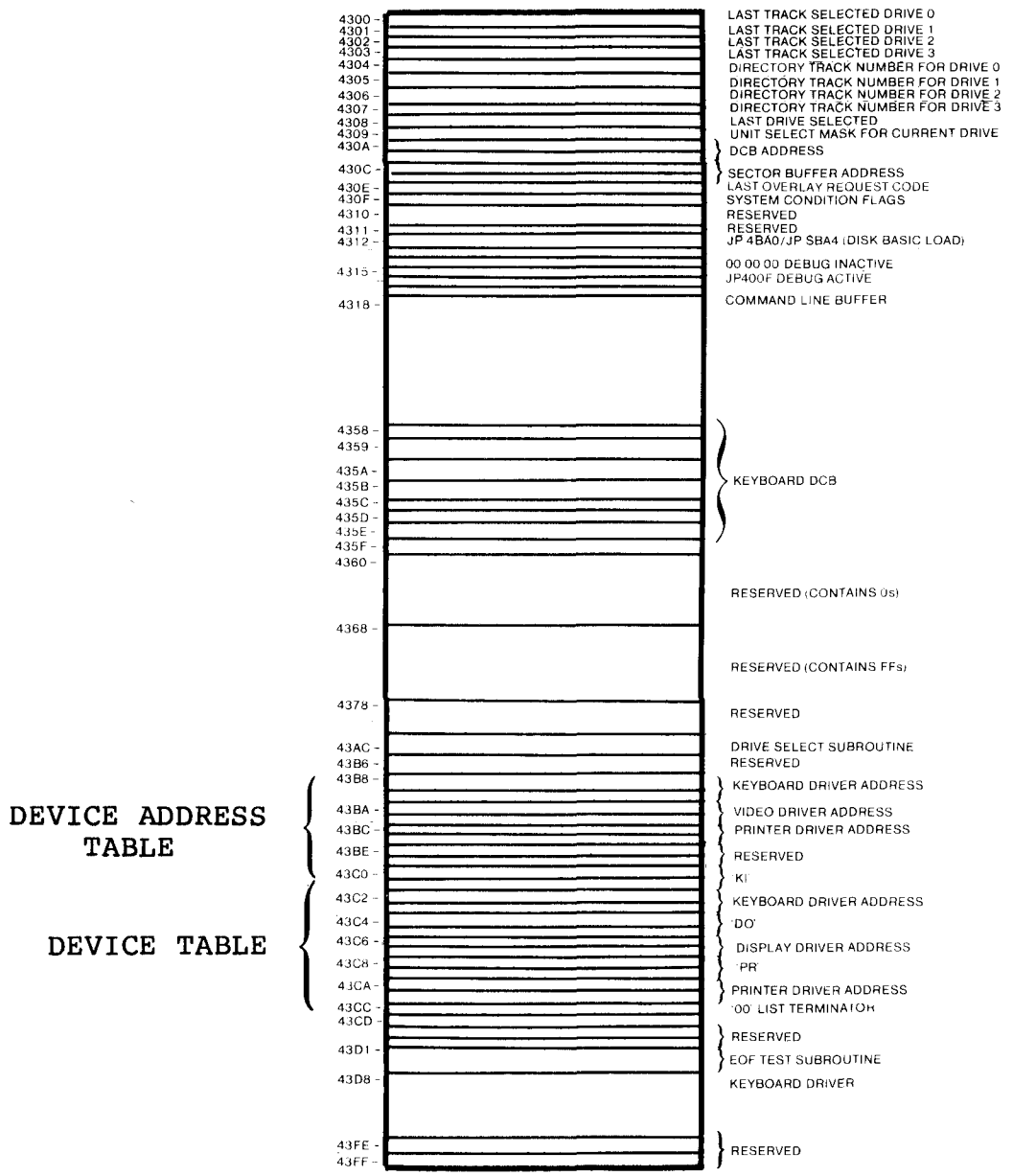
Appendix I

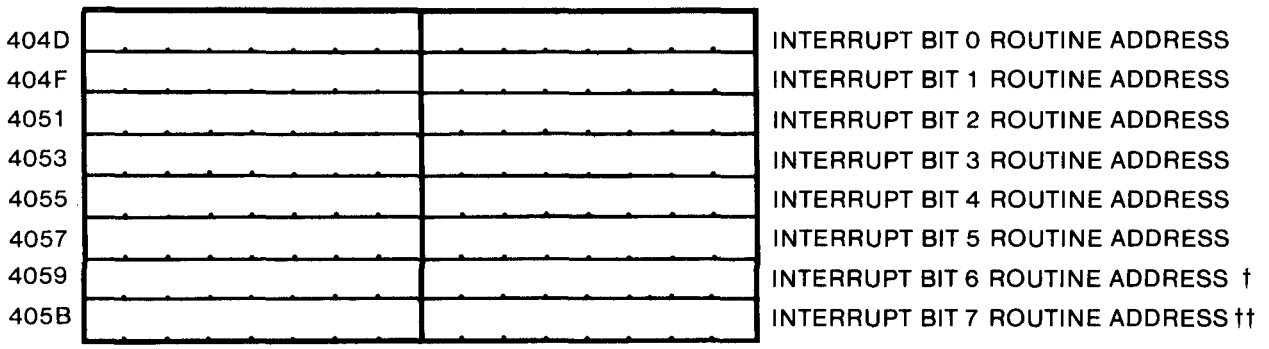
APPENDIX I
DATA STRUCTURES

SYS0 MEMORY MAP

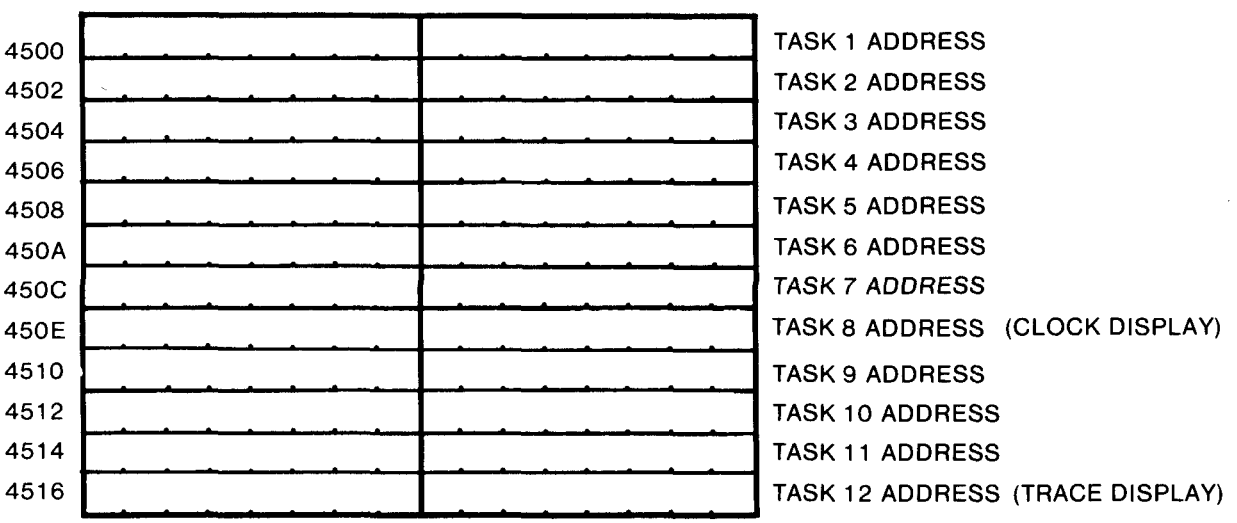


DESCRIPTION OF 4300 REGION

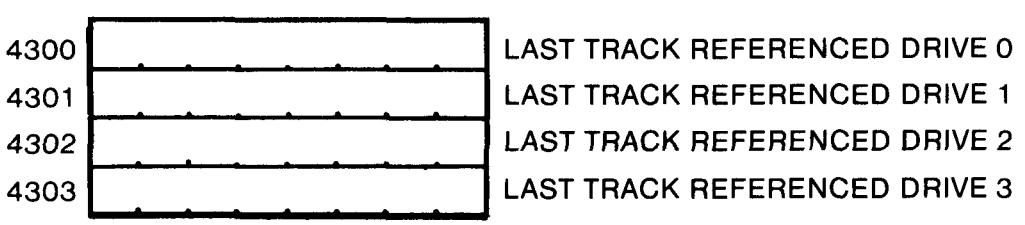




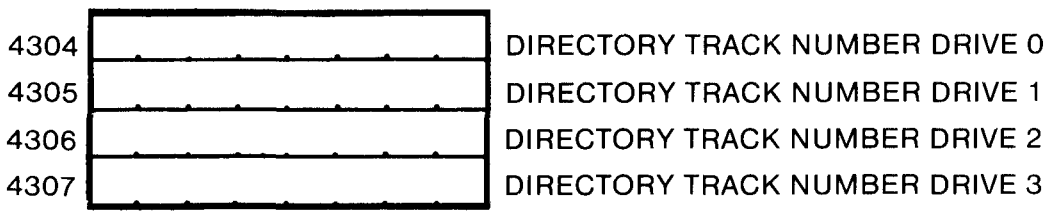
INTERRUPT SERVICE LIST



INTERRUPT TASK LIST ADDRESSES



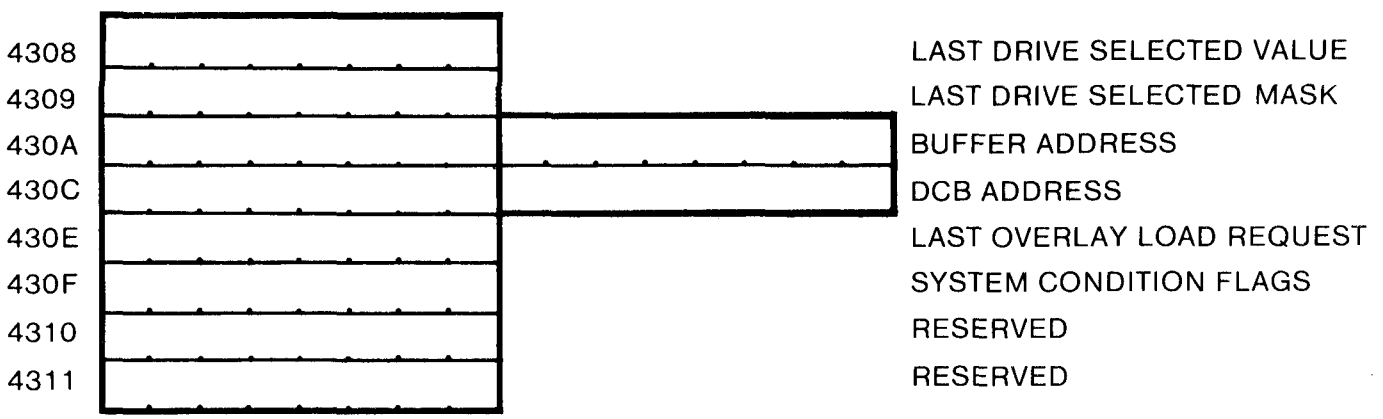
TRACK HISTORY TABLE



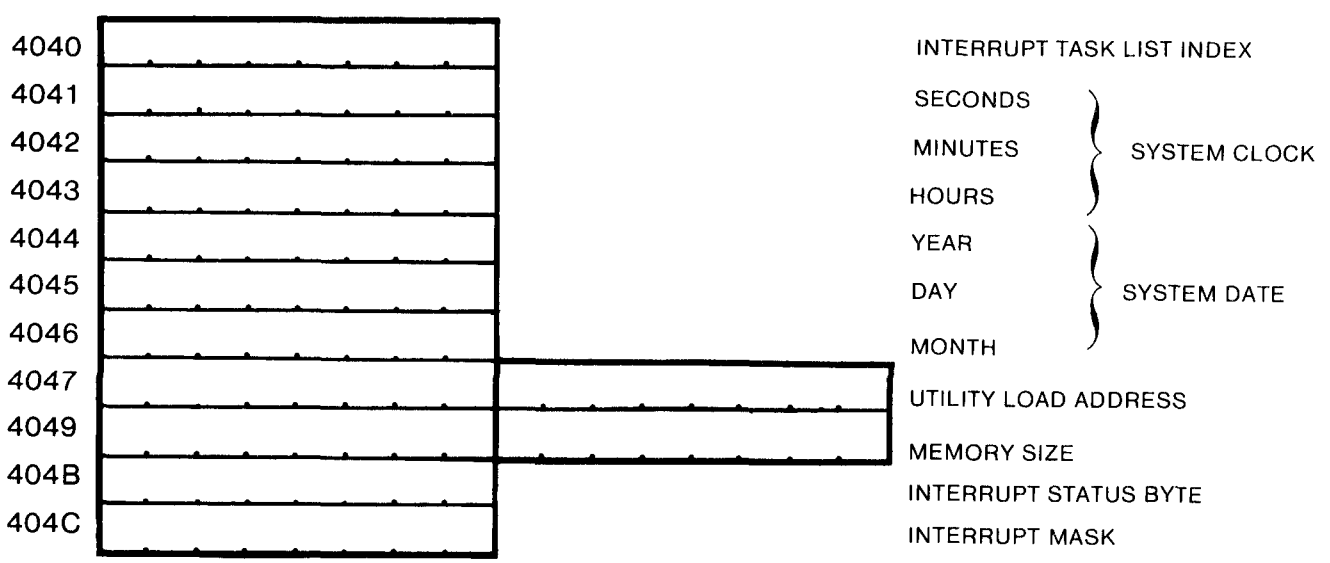
DIRECTORY TRACK LIST

† Disk interrupt

†† Clock interupt

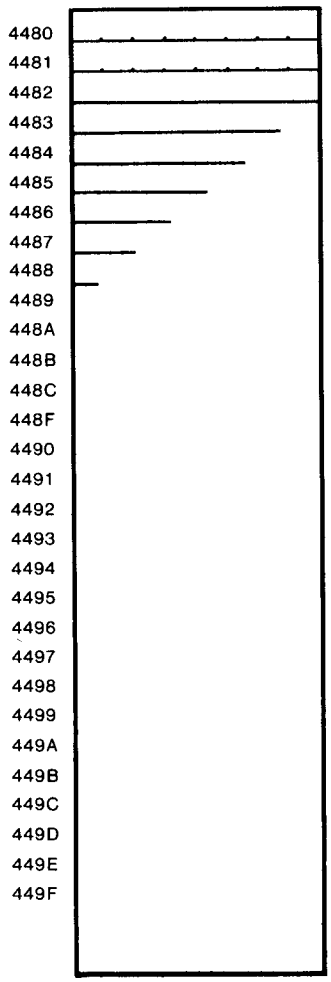


UNIT FLAGS AND ADDRESSES

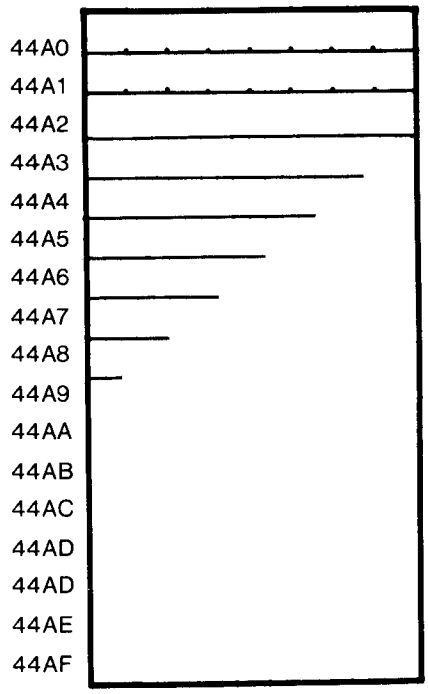


LOCAL STORAGE FOR NUCLEUS

SYSTEM DCB (LONG)



SYSTEM DCB (SHORT)



SYSØ ENTRY POINTS

Entry points for SYSØ are divided into two groups. The first group consists of nucleus subroutines which are entered indirectly through calls to LEVEL II ROM addresses.

Group 2 consists of nucleus addresses which are called directly. There is some overlap between Group 1 and Group 2 functions. For example, the Group 1 call to 402D is equivalent to the Group 2 call to 4400; however, a differant calling sequence is used.

SYSØ ENTRY POINTS (GROUP 1)

Address	Contents	Calling Sequence	Description
400C	JP 4BA2H	CALL LOAD . . . LOAD LD A,FCODE RST 28H	Loads and executes the overlay processor for the function code in the A-register . Note, the return address saved by the RST 28H instruction is lost during processing at 4BA2; therefore, this calling sequence must be used from a CALL'ed subroutine.
400F	JP 44B4H	RST 30H	Loads and executes the DEBUG utility. Control will be returned to the next instruction.
4012	JP 4518H	RST 38H	Executes the nucleus interrupt service routine. If no interrupts are pending at the time of entry, control returns immediately to the next instruction.
402D	JP 4400H	CALL SYS11 . . . SYS11 JP 402DH	Loads and executes SYS1/SYS with option code Ø1.
4030	LD A,ØA3H RST 28H	CALL SYS11 . . . SYS12 JP 4030H	Loads and executes SYS1/SYS with option code Ø2.

4033	JP 44BB	LD DE,DCB
		CALL 13H
		
		or
		
		LD A,VALUE
		LD DE,DCB
		CALL 1BH
			Read next character from disk file. DE contains the DCB address for an OPEN'ed file. Data character is retruned in A-register with ZERO status if no error. CARRY indicates end of file. A *NON-ZERO status is accompanied with a TRSDOS* error code in the A-register; uses all registers except BC.
			
			Write the character in the A-register to the OPEN'ed disk file specified by the DCB address in DE. On return, a NON-ZERO status indicates an error. The A register will contain a TRSDOS error code; uses all registers except BC.

SYSØ ENTRY POINTS (GROUP 2)

Address	Contents	Calling Sequence		Description
4400	LD A,93H RST 28H	CALL	4400H	Load and execute SYS1/SYS, option code Ø1.
4405	LD A,ØB3H RST 28H	CALL	4405H	Load and execute SYS1/SYS, option code Ø3.
4409	JP 44BØH	LD CALL	A,ECODE 4409H	Loads and executes SYS4/SYS to process the error code in the A-register.
440D	JP 44B4H	CALL	440DH	Loads and executes the DEBUG utility.
4410	JP 4596H	LD LD CALL	A,INDEX DE, ADDR 4410H	Stores the address given in the DE in the Interrupt Task List table for the index specified in the A-register.
4413	JP 4593H	LD CALL	A,INDEX 4413H	Stores the address of a RET instruction in the Interrupt Task List table for the index specified in the A-register.
4416	JP 45A5H	LD CALL	DE,ADDR 4416H	Replace first pointer in Interrupt Task List table for current interrupt process with address in DE.
4419	JP 458EH	CALL	4419H	Replace first pointer in Interrupt Task List table for current interrupt process with address of a RET instruction.
441C	LD A,ØC3H RST 28H	LD LD CALL	DE,SOURCE HL,DCB 441CH	Loads and executes SYS1/SYS, option code Ø4. Copies and validates file name specified by DE into DCB area given in HL.
4420	LD A,ØA4H RST 28H			TRSDOS INIT call. See TRSDOS & DISK BASIC REFERENCE MANUAL (RS Catalog No. 26-2104) pages 6 - 8.
4424	LD A,94H RST 28H			TRSDOS OPEN Call. See TRSDOS & DISK BASIC REFERENCE MANUAL, pages 6 - 9.
4428	LD A,95H RST 28H			TRSDOS CLOSE Call. See TRSDOS & DISK BASIC REFERENCE MANUAL, pages 6 - 11.
442C	LD A,ØA5H RST 28H			-TRSDOS KILL Call. See TRSDOS & DISK BASIC REFERENCE MANUAL, pages 6 - 11.
4430	JP 4C16H	LD CALL	DE,FNAME 4430H	Opens and loads binary file specified by the address in DE. A successful load is indicated by a ZERO status. The transfer address is returned in the HL, all registers are used. The system sector buffer at 4200 is used during the load operation.
4433	JP 4CØ6H	LD CALL	DE, FNAME 4433H	Opens. loads, and executes the binary file specified by the address in DE. If the DEBUG flag is active, DEBUG is executed prior to entering the loaded module.
4436	JP 476DH			TRSDOS READ Call. See TRSDOS & DISK BASIC REFERENCE MANUAL, pages 6 - 9.

SYSØ ENTRY POINTS (GROUP 2)

Address	Contents	Calling Sequence	Description
4439	JP 478BH		TRSDOS WRITE Call. See TRSDOS & DISK BASIC REFERENCE MANUAL, pages 6 - 10.
443C	JP 47A8H		TRSDOS VERIFY Call. See TRSDOS & DISK BASIC REFERENCE MANUAL, pages 6 - 10.
443F	JP 4756H	LD DE, DCB CALL 443FH	Rewinds the opened disk file specified by DE.
4442	JP 47ØØH		TRSDOS POSN call. See TRSDOS & DISK BASIC REFERENCE MANUAL, pages 6 - 9.
4445	JP 4737H	LD DE, DCB CALL 4445H	Backspace file 1 record. DE contains DCB address of the OPEN'ed file to be backspaced. All registers except AF are preserved.
4448	JP 475FH	LD DE, DCB CALL 4448H	Position DCB pointers to end of file for the file specafied by DCB. File must be OPEN'ed. All registers except AF are preserved.
4467	JP 44CFH	LD HL, M56 CALL 4467H	Display message specified by the address in HL on the video. Message must be terminated by an Ø3, or ØD. Uses A-register.
446A	JP 44DFH	LD HL, M5G CALL 446AH	Sends message specified by address in HL to printer. Message must be terminated by Ø3 or ØD. Uses all registers.
446D	JP 4CB7H	LD HL, aTIME CALL 446DH	Returns the system clock in ASCII to the address specified in HL. Format of the total time returned is: HH:MM:SS Uses all registers
4470	JP 4CD2H	LD HL, ADATE CALL 4470H	Returns the system date in ASCII to the address specified in HL. Format of the date returned is: MM/DD/YY Uses all registers.

4473	LD A,ØD3H
	RST 28H
		LD DE,FNAME
		LD HL,SUFFIX
		CALL 4473H
			Adds suffix specified by HL onto file name, specified by DE by calling SYS1/SYS option Ø5. Uses all registers.

4476	LD A,ØE3H
	RST 28H
		LD HL, CSTRING
		LD DE, ØLIST
		CALL 4476H
			Calls SYS1/SYS option code Ø6 to parse the command string specified by HL using the options list specified by DE. Uses all registers

44BØ	PUSH AF DD A,86H RST 28H	LD A, ERRCD CALL 44BØH	Calls SYS4/SYs to process the error code in the A regaster. Uses all registers.
44B4	PUSH AF LD A,87H RST 28	CALL 44B4H	Loads and executes the DEBUG utility.

SYS1 DATA STRUCTURES

SYS1 COMMAND LIST

```
4EA4            DEFM            'BASIC 2'
4EAA            DEFW            514CH
43AC            DEFM            'DEBUG'
4EB2            DEFW            5162H
4EB4            DEFM            'TRACE'
4EBA            DEFW            5191H
4EBC            DEFB            Ø
```

TRSDOS COMMAND LIST

```
4EBD            DEFM            'APPEND'
4EC3            DEFW            4E9ØH
4EC5            DEFM            'ATTRIB'
4ECA            DEFW            4E9ØH
4ECD            DEFM            'AUTO'
4ED3            DEFW            4E9ØH
4ED5            DEFM            'CLOCK'
4EDB            DEFW            4E9ØH
4EDD            DEFM            'COPY'
4EE3            DEFW            4E9ØH
4EE5            DEFM            'DATE'
4EEB            DEFW            4E9ØH
4EED            DEFM            'DEVICE'
4EF3            DEFW            4E9ØH
4EF5            DEFM            'DIR'
4EFB            DEFW            4E9ØH
```

TRSDOS COMMAND LIST

```
4EFD            DEFM            'DUMP'
4FØ3            DEFW            4E9ØH
4FØ5            DEFM            'FREE'
4FØB            DEFW            4E9ØH
4FØD            DEFM            'KILL'
4F13            DEFW            4E9ØH
4F15            DEFM            'LIB'
4F1B            DEFW            4E9ØH
4F1D            DEFM            'LIST'
4F23            DEFW            4E9ØH
4F25            DEFM            'LOAD'
4F2B            DEFW            4E9ØH
4F2D            DEFM            'PRINT'
4F33            DEFW            4E9ØH
4F35            DEFM            'PROT'
4F3B            DEFW            4E9ØH
4F3D            DEFM            'RENAME'
4F43            DEFW            4E9ØH
4F45            DEFM            'TIME'
4F4B            DEFW            4E9ØH
4F4D            DEFM            'VERIFY'
4F53            DEFW            4E9Ø
4F55            DEFB            Ø
```

PARAMETER SEPARATOR LIST

```
4FA7            DEFM            'TO'
4FAD            DEFW            Ø
4FAF            DEFM            'ON'
4FB6            DEFW            Ø
4FB7            DEFM            'OVER'
4FBD            DEFW            Ø
4FBF            DEFB            Ø
```

SYSTEM MESSAGES AND FILE SUFFIXES

```
51B5            DEFM            'END'
51B8            DEFB            ØAH
51B9            DEFM            'DOS READY'
51C2            DEFB            ØDH
51C3            DEFM            'WHAT?'
51C8            DEFB            ØDH
```

SYS4 DATA STRUCTURES

ERROR MESSAGES

ADDRESS		MESSAGE
4F36	DEFB	ØAH
4F37	DEFM	'*** ERRCOD * XX,'
4F46	DEFB	Ø3
4F47	DEFB	'***'
4F4A	DEFB	ØDH
4F4B	DEFB	ØC6@
4F4C	DEFM	'<DEVICE = *XX>
4F58	DEFB	ØDH
4F59	DEFB	C5H
4F5A	DEFM	'<FILE = NNNNNNNN/EEE:D>
4F6F	DEFB	ØDH
4F7Ø	DEFB	C3H
4F71	DEFM	'REFERENCED AT X'
4F8Ø	DEFB	27H
4F81	DEFB	Ø3H
4F82	DEFB	27H
4F83	DEFB	ØDH

ERROR CODE INDEX LIST

ADDRESS		LSB OF WORD INDEX LIST ADDR.	ERROR CODE
4F84	DEFB	5BH	ØØ
4F85	DEFB	5DH	Ø1
4F86	DEFB	62H	Ø2
4F87	DEFB	66H	Ø3
4F88	DEFB	6AH	Ø4
4F89	DEFB	6EH	Ø5
4F8A	DEFB	74H	Ø6
4F8B	DEFB	79H	Ø7
4F8C	DEFB	7EH	Ø8
4F8D	DEFB	81H	Ø9
4F8E	DEFB	86H	1Ø
4F8F	DEFB	8AH	11
4F9Ø	DEFB	8EH	12
4F91	DEFB	92H	13
4F92	DEFB	98H	14
4F93	DEFB	9DH	15
4F94	DEFB	AØH	16
4F95	DEFB	A4H	17
4F96	DEFB	A7H	18
4F97	DEFB	AAH	19
4F98	DEFB	ADH	2Ø
4F99	DEFB	BØH	21
4F9A	DEFB	B3H	22
4F9B	DEFB	B6H	23
4F9C	DEFB	B9H	24
4F9D	DEFB	BDH	25
4F9E	DEFB	CØH	26
4F9F	DEFB	C3H	27
4FAØ	DEFB	C6H	28
4FA1	DEFB	CAH	29
4FA2	DEFB	CFH	3Ø
4FA3	DEFB	D3H	31
4FA4	DEFB	D6H	32
4FA5	DEFB	D9H	33
4FA6	DEFB	DDH	34
4FA7	DEFB	E1H	35

ERROR CODE INDEX LIST

ADDRESS		LSB OF WORD INDEX LIST ADDR.	ERROR CODE	
4FA8	DEFB	E3H	36	
4FA9	DEFB	E8H	37	
4FAA	DEFB	EDH	38	
4FAB	DEFB	F0H	39	
4FAC	DEFB	F0H	40	
4FAD	DEFB	F0H	41	
4FAE	DEFB	F0H	42	
4FAF	DEFB	F0H	43	Not Implemented
.	DEFB	.	.	
.	DEFB	.	.	
.	DEFB	.	.	
4FC3	DEFB	.	63	

WORD ADDRESS LIST

ADDRESS		WORD ADDRESS	WORD
4FC4	DEFW	5030H	NO
4FC6	DEFW	5032H	ERROR
4FC8	DEFW	5037H	FORMAT
4FCA	DEFW	503DH	PARITY
4FCC	DEFW	5043H	DURING
4FCE	DEFW	5049H	HEADER
4FD0	DEFW	504FH	DATA
4FD2	DEFW	5053H	SEEK
4FD4	DEFW	5057H	READ
4FD6	DEFW	505BH	WRITE
4FD8	DEFW	5060H	LOST
4FDA	DEFW	5064H	NOT
4FDC	DEFW	5067H	ATTEMPTED TO
4FDE	DEFW	5073H	LOCKED/DELETED
4FE0	DEFW	5081H	SYSTEM
4FE2	DEFW	5087H	DIRECTORY
4FE4	DEFW	5090H	MEMORY
4FE6	DEFW	5096H	ON
4FE8	DEFW	5098H	DISK
4FEB	DEFW	509CH	DISKETTE
4FEC	DUFW	50A4H	FAULT
4FEE	DEFW	50A9H	PROTECTED
4FF0	DEFW	50B2H	ILLEGAL
4FF2	DEFW	50B9H	LOGICAL
4FF4	DEFW	50C0H	NUMBER

WORD ADDRESS LIST

ADDRESS		WORD ADDRESS	WORD
4FF6	DEFW	50C6H	FILE
4FF8	DEFW	50CAH	RECORD
4FFA	DEFW	50D0H	END
4FFC	DEFW	50D3H	OF
4FFE	DEFW	50D5H	OUT
5000	DEFW	50D8H	RANGE
5002	DEFW	50DDH	ENCOUNTERED
5004	DEFW	50E8H	CODE
5006	DEFW	50ECH	GAT
5008	DEFW	50EFH	HIT
500A	DEFW	50F2H	OVERLAY
500C	DEFW	50F9H	UNKNOWN
500E	DEFW	5100H	LOAD
5010	DEFW	5104H	SPACE
5012	DEFW	5109H	ONLY
5014	DEFW	510DH	NAME
5016	DEFW	5111H	DEVICE
5018	DEFW	5117H	FORMAT
501A	DEFW	511DH	FOUND
501C	DEFW	5122H	IN
501E	DEFW	5124H	ACCESS
5020	DEFW	512AH	FULL
5022	DEFW	512EH	DRIVE
5024	DEFW	5133H	DENIED
5026	DEFW	5139H	PROGRAM
5028	DEFW	5140H	AVAILABLE
502A	DEFW	5149H	CAN'T EXTEND
502C	DEFW	5157H	OPEN
502E	DEFW	515BH	ADDRESS OF MESSAGE LIST

WORD LIST

ADDRESS	WORD ADDRESS	WORD
5030	DEFM	'NO'
5032	DEFM	'ERROR'
5036	DEFM	'FORMAT'
503C	DEFM	'PARITY'
5043	DEFM	'DURING'
5048	DEFM	'HEADER'
504F	DEFM	'DATA'
5053	DEFM	'SEEK'
5057	DEVM	'READ'
505B	DEFM	'WRITE'
5060	DEFM	'LOST'
5064	DEFM	'NOT'
5067	DEFM	'ATTEMPTED TO'
5073	DEFM	'LOCKED'
5078	DEFM	'/DELETED'
5081	DEFM	'SYSTEM'
5087	DEFM	'DIRECTORY'
5090	DEFM	'MEMORY'
5096	DEFM	'ON'
5098	DEFM	'DISK'
509C	DEFM	'DISKETTE'
50A4	DEFM	'FAULT'
50A9	DEFM	'PROTECTED'
50B2	DEFM	'ILLEGAL'
50B9	DEFM	'LOGICAL'
50C0	DEFM	'NUMBER'
50C6	DEFM	'FILE'
50CA	DEFM	'RECORD'

WORD LIST

ADDRESS	WORD ADDRESS	WORD
50D0	DEFM	'END'
50D3	DEFM	'OF'
50D5	DEFM	'OUT'
50D8	DEFM	'RANGE'
50DD	DEFM	'ENCOUNTERED'
50E8	DEFM	'CODE'
50EC	DEFM	'GAT'
50EF	DEFM	'HIT'
50F2	DEFM	'OVERLAY'
50F9	DEFM	'UNKNOWN'
50FF	DEFM	'LOAD'
5104	DEFM	'SPACE'
5109	DEFM	'ONLY'
510D	DEFM	'NAME'
510F	DEFM	'DEVICE'
5117	DEFM	'FORMAT'
511D	DEFM	'FOUND'
5121	DEFM	'IN'
5124	DEFM	'ACCESS'
512A	DEFM	'FULL'
512E	DEFM	'DRIVE'
5133	DEFM	'DENIED'
5139	DEFM	'PROGRAM'
5140	DEFM	'AVAILABLE'
5149	DUFM	'-CAN'T EXTEND'
5157	DEFM	'OPEN'

MESSAGE LIST

ADDRUSS	WORD	INDEX	MESSAGE
515B	DEFB	Ø1H	NO ERROR
515C	DEFB	82H	
515D	DEFB	Ø4H	PARITY ERROR DURING HEADER READ
515E	DEFB	Ø2H	
515F	DEFB	Ø5H	
516Ø	DEFB	Ø6H	
5161	DEFB	89H	
5162	DEFB	Ø8H	SEEK ERR_R DURING READ
5163	DEFB	Ø2H	
5164	DEFB	Ø5H	
5165	DEFB	89H	
5166	DEFB	ØBH	LOST DATA DURING READ
5167	DEFB	Ø7H	
5168	DEFB	Ø5H	
5169	DEFB	89H	
516A	DEFB	Ø4H	PARITY ERROR DURING READ
516B	DEFB	Ø2H	
596C	DEFB	Ø5H	
516D	DEFB	89H	
516E	DEFB	Ø7H	DATA RECORD NOT FOUND DURING READ
516F	DEFB	1BH	
517Ø	DEFB	ØCH	
5171	DEFB	2CH	
5172	DEFB	Ø5H	
5173	DEFB	89H	
5174	DEFB	ØDH	ATTEMPTED TO READ LOCKED RECORDS
5175	DEFB	Ø9H	
5176	DEFB	ØEH	
5177	LEFB	Ø7H	
5178	DEFB	9BH	

MESSAGE LIST

ADDRESS	WORD	INDEX	MESSAGE
5179	DEFB	ØDH	ATTEMPTED TO READ DELETED RECORD
517A	DEFB	Ø9H	
517B	DEFB	ØFH	
517B	DEFB	Ø7H	
517D	DEFB	9BH	
517E	DEFB	2AH	DEVICE NOT AVAILABLE
517F	DEFB	ØCH	
518Ø	DEFB	F3H	
5181	DEFB	Ø4H	PARITY ERROR DURING HEADER WRITE
5182	DEFB	Ø2H	
5183	DEFB	Ø5H	
5184	DEFB	Ø6H	
5185	DEFB	8AH	
5186	DEFB	Ø8H	SEEK ERROR DURING WRITE
5187	DEFB	Ø2H	
5188	DEFB	Ø5H	
5189	DEFB	8AH	
518A	DEFB	ØBH	LOST DATA DURING WRITE
518B	DEFB	Ø7H	
518C	DEFB	Ø5H	
518D	DEFB	8AH	
518E	DEFB	Ø4H	PARITY ERROR DURING WRITE
518F	DEFB	Ø2H	
519Ø	DEFB	Ø5H	
5191	DEFB	8AH	

MESSAGE LIST

ADDRESS	WORD	INDEX	MESSAGE
5192	DEFB	07H	DATA RECORD NOT FOUND DURING WRITE
5193	DEFB	1BH	
5194	DEFB	0CH	
5195	DEFB	2CH	
5196	DEFB	05H	
5197	DEFB	8AH	
5198	DEFB	0AH	WRITE FAULT MEMORY DISK DEVICE
5199	DEFB	15H	
519A	DEFB	12H	
519B	DEFB	B0H	
519D	DEFB	0AH	WRITE PROTECTED DISKETTE
519E	DEFB	16H	
519F	DEFB	94H	
51A0	DEFB	17H	ILLEGAL LOGICAL FILE NUMBER
51A1	DEFB	18H	
51A2	DEFB	1AH	
51A3	DEFB	99H	
51A4	DEFB	10H	SYSTEM READ ERROR
51A5	DEFB	09H	
51A6	DEFB	82H	
51A7	DEFB	10A	SYSTEM WRITE ERROR
51A8	DEFB	0AH	
51A9	DEFB	82H	
51AA	DEFB	17H	ILLEGAL FILE NAME
51AB	DEFB	1AH	
51AC	DEFB	E9H	

MESSAGE LIST

ADDRESS	WORD	INDEX	MESSAGE
51AD	DEFB	22H	GAT READ ERROR
51AE	DEFB	Ø9H	
51AF	DEFB	82H	
51BØ	DEFB	22H	GAT WRITE ERROR
51Bl	DEFB	ØAH	
51B2	DEFB	82H	
51B3	DEFB	23H	HIT READ ERROR
51B4	DEFB	Ø9H	
51B5	DEFB	82H	
51B6	DEFR	23H	HIT WRITE ERROR
51B7	DEFB	ØAH	
51B9	DEFB	82H	
51B9	DEFB	lAH	FILE NOT IN SYSTEM
51BA	DEFB	ØCH	
51BB	DEFB	2DH	
51BC	DEFB	DØH	
51BD	DEFB	lAH	FILE ACCESS DENIED
51BE	DEFB	2EH	
51BF	DEFB	FlH	
51CØ	DEFB	lØH	SYSTEM SPACE FULL
51Cl	DEFB	27H	
51C2	DEFB	EFH	

MESSAGE LIST

ADDRESS	WORD INDEX		MESSAGE
51C3	DEFB	13H	DISK SPACE FULL
51C4	DEFB	27H	
51C5	DEFB	AFH	
51C6	DEFB	1CH	END OF FILE ENCOUNTERED
51C7	DEFB	1DH	
51C8	DEFB	1AH	
51C9	DEFB	AOH	
51CA	DEFB	1BH	RECORD NUMBER OUT OF RANGE
51CB	DEFB	19H	
51CC	DEFB	1EH	
51CD	DEFB	1DH	
51CE	DEFB	9FH	
51CF	DEFB	1ØH	SYSTEM FULL - CAN'T EXTEND FILE
51DØ	DEFB	2FH	
51D1	DEFB	34H	
51D2	DEFB	9AH	
51D3	DEFB	32H	PROGRAM NOT FOUND
51D4	DEFB	ØCH	
51D5	DEFB	ACH	
51D6	DEFB	17H	ILLEGAL DRIVE NUMBER
51D7	DEFB	3ØH	
51D8	DEFB	D9	

MESSAGE LIST

ADDRESS	WORD INDEX	MESSAGE
51D9	DEFB 01H	NO DEVICE SPACE AVAILABLE
51DA	DEFB 2AH	
51DB	DEFB 27H	
51DC	DEFB F3H	
51DD	DEFB 26H	LOAD FILE FORMAT ERROR
51DE	DEFB 1AH	
51DF	DEFB 2BH	
51E0	DEFB 82H	
51E1	DEFB 11H	DIRECTORY FAULT
51E2	DEFB 95H	
51E3	DEFB 0DH	ATTEMPTED TO LOAD READ ONLY FILE
51E4	DEFB 26H	
51E5	DEFB 09H	
51E6	DEFB 28H	
51E7	DEFB 91H	
51E8	DEFB 17H	ILLEGAL ACCESS ATTEMPTED TO FILE
51E9	DEFB 2EH	
51EA	DEFB 0DH	
51EB	DEFB 16H	
51ES	DEFB 9AH	
51ED	DEFB 1AH	FILE NOT OPEN
51EE	DEFB 0CH	
51EF	DEFB F5H	
51F0	DEFB 25H	UNKNOWN ERROR CODE
51F1	DEFB 02H	
51F2	DEFB A1H	

SYS5 DATA STRUCTURES

DISPLAY HEADERS

ADDRESS		TEXT
4F54	DEFM	"AF BC DE HL AF'DE'HL'IX IY SP "
4F78	DEFM	'SZIHIPNC'

The code/length byte in the instruction group classification tables has the following format:

Code: A 4-bit code specifying the reference address for the next I-fetch. The codes, and their interpretations are as follows:

- Ø - Next instruction address follows current instruction.
- 1 - Next instruction address is indirect through the HL. JP (HL).
- 2 - Next instruction address is conditionally the operand of the current instruction. JP C,XXXX.
- 3 - Next instruction address is conditionally the operand of the current instruction. DJNZ XXXX.
- 4 - Next instruction address is the operand of the current instruction. JP XXXX.
- 5 - Next instruction address is conditional and/or indirect through the stack. RET, RET NZ.
- 6 - Next instruction address will be entered via a CALL. CALL XXXX, or CALL NZ, XXXX.

Length: - A 4 Bit value giving the length in bytes of the current instruction.

GROUP 1 INSTRUCTIONS

Address		Mask, Op code, Code/length
50E0	DEFB	C7H,C0H,51H
50E3	DEFB	FFH,C9H,51H
50E6	DEFB	FFH,E9H,11H
50E9	DEFB	CFH,0IH,03H
50EC	DEFB	E7H,22H,03H
50EF	DEFB	C7H,C2H,43H
50F2	DEFB	FFH,C3H,43H
50F5	DEFB	C7H,C4H,63H
50F8	DEFB	FFH,CDH,63H
50FB	DEFB	C7H,06H,02H
50FE	DEFB	F7H,D3H,02H
5101	DEFB	C7H,C6H,02H
5104	DEFB	FFH,CBH,02H
5107	DEFB	F7H,10H,32H
510A	DEFB	E7H,20H,32H
510D	DEFB	01H

GROUP 2 INSTRUCTIONS

Address		Mask, Op Code, Code/length
510E	DEFB	C7H,43H,04H
5111	DEFB	F7H,45H,52H
5114	DEFB	02H

GROUP 3 INSTRUCTIONS

Address		Mask, OP Code, Code/length
5115	DEFB	FEH,34H,03H
5118	DEFB	C0H,40H,03H
511B	DEFB	C0H,80H,03M
511E	DEFB	FFH,21H,04H
5121	DEFB	FFH,22H,04H
5124	DEFB	FFH,2AH,04H
5127	DEFB	FFH,36H,04H
512A	DEFB	FFH,CBH,04H
512D	DEFB	FFH,E9H,22H
5130	DEFB	02H

SYS6 DATA STRUCTURES

ERROR MESSAGES AND PROMPTS

ADDRESS		MESSAGE
525A	DEFM	'RESERVED COMMAND'
5274	DEFM	'NOTHING DONE'
52DF	DEFM	'BAD FORMAT'
5477	DEFM	'INVALID COMMAND DURING PROGRAM CHAINING'
54A0	DEFM	'MASTER PASSWORD ? '
54B9	DEFM	'NEW MASTER PASSWORD ? '
54D2	DEFM	'INVALID MASTER PASSWORD'
5592	DEFM	'FILE SPEC REQUIRED'
55A6	DEFM	'DEVICE SPEC REQUIRED'
58C5	DEFM	'END LESS THAN START'
58DA	DEFM	'START LESS THAN X '7000''
58F2	DEFM	'CIM'
5C43	DEFM	'ATTRIBUTE SPECIFICATION ERROR'
5E85	DEFM	'FILE DIRECTORY --- DRIVE X'
5EA1	DEFM	'NNNNNNNN -- MM/DD/YY'
5EB7	DEFM	'LRL= / END= / SIZE = GRANS'
5F62	DEFM	'DRIVE X-- '
5F6E	DEFM	'NNNNNNNN/ /MM/DD/YY / FILES, GRANS'
6031	DEFM	'DRIVE SPECIFICATION ILLEGAL'
6057	DEFM	'DUPLICATE FILE NAME'

PARAMETER TEXT STRINGS

ADDRESS		TEXT STRING
54EB	DEFM	'PW '
54F1	DEFW	53D9H
54F3	DEFM	'LOCK '
54F9	DEFW	53F9H
54FB	DEFM	'UNLOCK'
5501	DEFW	5403H
5503	DEFB	00H
5C61	DEFM	'EXREWR'
5E6A	DEFM	'A '
5E70	DEFW	5DB9H
5E72	DEFM	'I '
5E78	DEFW	5D0BH
5E7A	DEFM	'S '
5E80	DEFW	5CFDH
5382	DEFB	00
58F5	DEFM	'START '
58FB	DEFW	5868H
58FD	DEFM	'END '
5903	DEFW	586EH
5905	DEFM	'TRA '
590B	DEFW	58ABH
590D	DEFB	0
598B	DEFM	'LINE '
5991	DEFW	5954H

LOCAL STORAGE AREAS

ADDRESS		SIZE	COMMENT
5551	DEFS	20H	DCB
5571	DEFS	20H	DCB
5E67	DEFS	03H	EOF RECORD NUMBER
590E	DEFS	06H	BUFFER AREA FOR FILE NAME

SYSØ CROSS REFERENCES

SYSØ - SYS1 CROSS REFERENCE

SYSØ ADDRESS	SYS1 REFERENCE ADDRESS
4000	514F
400F	516D
402F	516D
402D	4E4E
4030	4E3D, 4E8D, 517E
4031	517A
430F	4E28, 4E72, 4E77, 4E90, 4E98, 4E9D, 5165
4315	4E31, 5175, 5187
4316	5172
4318	4E23
4410	5199
4433	4E81
4467	4E43, 4E8A
4480	4E52
4CD9	5194

SYSØ - SYS2 CROSS REFERENCE

SYSØ ADDRESS	SYS2 REFERENCE ADDRESS
4040	50AB
45CF	514B
4600	5102
4896	4E12, 4ED8
4ABE	4F77, 4F92
4AC1	4E6D, 4F17, 4F6F, 4F96
4AD6	4F46, 4F8E, 4FA3
4B35	5138
4B55	5130, 5143
4D00	5135, 5148

SYSØ - SYS3 CROSS REFERENCE

SYSØ ADDRESS	SYS3 REFERENCE ADDRESS
4Ø4A	4EØ9
4E6F	5Ø7F
4878	4E71
4892	4E6D, 4FDØ
4AC1	4E7B, 4ECF, 4F5C, 4F81, 4FEF
4AD6	4EB1, 4F52, 4F6F, 5Ø17
4AFØ	4EF6, 4FEB
4BØ3	4F77, 5Ø26
4B55	5Ø68, 5Ø77
4B84	4EE2

SYSØ - SYS4 CROSS REFERENCE

SYSØ ADDRESS	SYS4 REFERENCE ADDRESS
4Ø3Ø	4E7Ø
43ØA	4E7E
43ØC	4FØC
4467	4E27, 4E77, 4E9B, 4FØ3, 4FØ9, 4F18
4552	4F7Ø
4AC1	4EC4

SYSØ - SYS5
CROSS REFERENCE

SYSØ ADDRESS	SYS5 REFERENCE ADDRESS
400F	4F99
4020	4E55, 4F48, 4FEC, 4FF9
405D	4E2F, 4ECB, 5144
405E	4E9A, 4EB4, 4ECF
4060	4FDB, 4FE1, 4FEF, 5165
4062	4F82, 5092
4063	4EB1, 4EC2, 4F2F, 4F4B
4065	4E17, 4EDA, 5049
406B	4FC6, 50D8
4075	50CC
4077	50D3
4079	4E1F, 4E4B, 50B7
407A	4FA4
407B	4E23, 4E2C, 4F8A, 4FC2, 505E
4315	4FA1
4316	4F9C
43C7	510D

SYSØ - SYS6
CROSS REFERENCE

SYSØ ADDRESS	SYS6 REFERENCE ADDRESS
402D	52Ø1, 5337, 5367, 54Ø9, 5451, 5512, 57B3 57E2, 58BB, 5929, 598Ø, 5985, 59A6, 59E3 59E8, 5C36, 5DD8, 5F5F
4Ø3Ø	5257, 5271, 52D9, 53D5, 5474, 55Ø4, 5549, 58C1, 5979, 59D4, 5C3F, 5DDE, 6Ø2E, 6Ø54
4Ø41	52BC
4Ø44	52A8
42ØØ	5418, 5442, 5CA5
4296	53FF, 5D7Ø, 5D83
42DØ	5CAE
43ØF	536A, 5515
43CØ	531D
44Ø9	554E
441Ø	52DØ
4413	52CD
441C	57Ø3, 5712, 5752, 575B, 57EC, 5917, 5936, 5997, 59AF, 5BØ3, 5F9F, 5FB4
442Ø	5777, 5832
4424	53AC, 5723, 5731, 5769, 591D, 594D, 59CØ, 5B15 5EE8, 5FAB, 5FE4
4428	57A4, 57AD, 58B5
442C	5923
443Ø	59A3
4436	578Ø
4439	5793
443A	55ØF
4448	5254, 526E, 52D6, 53B2, 53D2, 53E2, 5471, 553D, 5546, 58BE, 5C3C, 5C9B, 5CC4, 5E38, 5F55, 6Ø2B, 6Ø51
4473	582A
4476	5395, 57FB, 5945, 5C95
46EF	5445
478B	55Ø7, 6ØCD
47A8	55ØC
485E	6ØD5
4AC1	5BFB, 5CCD, 5FF4
4AD6	5C3Ø, 6ØØ8
4AFØ	5285, 53BD, 5EF7
4BØ3	5296, 53EF
4B35	541B, 5CA8
4B55	5413, 5CAØ
4CA9	52C8
4DØØ	5B12, 5F1Ø
4D4Ø	5F34
4DCE	53C3, 53E8, 53F5
4DDØ	5EFD

SYS1 CROSS REFERENCE

SYS1 - SYS6
CROSS REFERENCE

SYS1 ADDRESS	SYS6 REFERENCE ADDRESS
4EBD	533A
51A0	52C5, 5504

SYS2 CROSS REFERENCE

SYS2 - SYS6
CROSS REFERENCE

SYS2 ADDRESS	SYS6 REFERENCE ADDRESS
5081	5BF0
509B	601A
50D1	546B, 5BB1, 5BC6
50FD	5EEC
512E	5F2E, 600E
5141	601F
5155	5B56, 5BAE, 5BC3, 5BE1
5157	5B73
515D	5FFF, 6017
5168	53B8, 53C6, 5BC9, 5C17
5169	5C1B
516A	5BB4, 5C27
516B	5C2B

Appendix II

Appendix II
Complete Systems

```
*****************************************************************
*                          SYS0/SYS                             *
*                       Nucleus Routine                         *
*                            for                                *
*                         TRSDOS                                *
*****************************************************************

*
*            ----  Entry Points Used by LEVEL II  ----
*
400C C3A24B     JP      4BA2H       Jump to system overlay loader.
400F C3B444     JP      44B4H       Make overlay call to load DEBUG
4012 C31845     JP      4518H       Jump to interrupt service routine
*
*          ----  Keyboard, Video, and Printer DCB's  -----
*
4015            DEFS    8           Keyboard DCB (loaded by LEVEL II)
401D            DEFS    8           Video DCB (loaded by LEVEL II)
4025            DEFS    8           Printer DCB (loaded by LEVEL II)
*
*              ----  Overlay Load Requests  ----
*
402D C30044     JP      4400H       Make overlay call to load
                                    SYS1/SYS req code 1
*
*  If DEBUG active, locations 4030 - 4032 contain CALL 400F (load DEBUG)
*
4030 3EA3       LD      A,0A3H      Code for SYS1/SYS option 2
4032 EF         RST     28H         Load and execute overlay
                                    Does not return in-line
4033 C3BB44     JP      44BBH       Enter driver. Called from LEVEL II
*
*          ----  Local Storage Areas for Nucleus  ----
*
4036            DEFS    7           Keyboard row history table
*
*
403D 00         DEFB    00          Video display byte used by LEVEL II
403E 22         DEFB    21H         Reserved.
403F 01         DEFB    01H         Reserved.
4040            DEFB    00          Clock interrupt index.
4041            DEFB    00          Seconds
4042            DEFB    00          Minutes
4043            DEFB    00          Hours
4047            DEFW    0000        Utility program load address
4049            DEFW    0000        Memory size.
404B            DEFB    00          Current interrupt status byte
404C            DEFB    00          Interrupt subroutine mask
*
*              ----  Interrupt task list  ----
*
404D            DEFW    4537H       Address of interrupt service
                                    routine for interrupt bit 0
404F            DEFW    4537H       Address for bit 1
4051            DEFW    4537H       Address for bit 2
4053            DEFW    4537H       Address for bit 3
4055            DEFW    4537H       Address for bit 4
4057            DEFW    4537H       Address for bit 5
4059            DEFW    4537H       Address for bit 6
405B            DEFW    4537H       Address for bit 7
*
*              ----  System Sector Buffer  ----
*
4200            DEFS    256         System sector buffer area.
*
*              ----  Track History Table  ----
*
4300            DEFB    11H         Last track addressed drive 0
4301            DEFB    11H         Last track addressed drive 1
4302            DEFB    11H         Last track addressed drive 2
4303            DEFB    11H         Last track addressed drive 3
*
*              ----  Directory Track List  ----
*
4304            DEFB    11H         Directory track number drive 0
4305            DEFB    11H         Directory track number drive 1
4306            DEFB    11H         Directory track number drive 2
4307            DEFB    11H         Directory track number drive 3
*
*             -----Unit Flags And Addresses----
*
4308            DEFB    00H         Last unit selected
4309            DEFB    01H         Select mask for last unit selected.
430A            DEFB    00          Buffer address, current operation
430B            DEFB    00
430C            DEFB    00          DCB address for current operation
430D            DEFB    00
430E            DEFB    02H         Last overlay load request
430F            DEFB    00          DEBUG load flag
4310            DEFB    00          Reserved
4311            DEFB    00          Reserved
*
*
*
4312 C3A04B     JP      4BA0H       Return with zero status
                                    JP    5BA4 if DISK BASIC loaded
*
*          4315-4317 Contains DEBUG FLAG if DEBUG active
*
4315            DEFB    0           Contains 0 if DEBUG (off)
                                    C3 if DEBUG (on)
4316            DEFW    0           Addr. of nucleus to load DEBUG
*
*      ----  Wait for Disk Controller to Become Not Busy  ----
*
4398 3AEC37     LD      A,(37ECH)   Get controller status.
439B CB47       BIT     00H,A       Test if controller busy.
439D C8         RET     Z           Exit if not busy.
439E 18F8       JR      4398H       Loop till not busy.

*
*              ----  Reselect Last Unit Selected  ----
*
43AC 3A0943     LD      A,(4309H)   Get mask for current unit
43AF 32E137     LD      (37E1H),A   Select unit
43B2 3AEC37     LD      A,(37ECH)   Fetch controller status
43B5 C9         RET                 Return to caller
*
*        ----  Decrement Count of Records in File by 1  ----
*
43D1 DD7E08     LD      A,(IX+08H)  Get EOF byte in last sector
43D4 B7         OR      A           Set status flag and
43D5 C8         RET     Z           return with zero status
                                    if file ends on sector boundary
43D6 0B         DEC     BC          Else decrement no. of records
43D7 C9         RET                 Then return to caller
*
*                    ----  Keyboard Driver  ----
*
43D8 213640     LD      HL,4036H    Row history buffer address
43DB 013801     LD      BC,3801H    Memory mapped address, first row
43DE 1600       LD      D,00H       Row index
43E0 0A         LD      A,(BC)      Fetch a row from keyboard
43E1 5F         LD      E,A         Save row bits
43E2 AE         XOR     (HL)        Compare with previous contents
43E3 73         LD      (HL),E      Save new row value
43E4 A3         AND     E           Look for identical match
43E5 2008       JR      NZ,43EFH    Jump if not identical match
43E7 14         INC     D           Identical value. Bump row index
43E8 2C         INC     L           Bump row history buffer address
43E9 CB01       RLC     C           Shift row adr. Gives 2, 4, 8 ...
43EB F2E043     JP      P,43E0H     Loop till 7 rows tested
43EE C9         RET                 No new character. Return
43EF 5F         LD      E,A         Save active row bits
43F0 C5         PUSH    BC          Save active row address
43F1 01DF04     LD      BC,04DFH    Delay value for 18 milliseconds
43F4 CD6000     CALL    0060H       Call delay routine
43F7 C1         POP     BC          Restore active row address
43F8 0A         LD      A,(BC)      Refetch row bits
43F9 A3         AND     E           Compare with previous row bits
43FA C8         RET     Z           Exit if same row not active
43FB C3FB03     JP      03FBH       Else, call LEVEL II driver to
                                    translate row value to ASCII value
43FE            NOP
43FF            NOP
*
*              ----  Overlay Load Requests  ----
*
4400 3E93       LD      A,93H       Overlay request code for SYS1/SYS
                                    option 10
4402 EF         RST     28H         Load and execute overlay
4403            NOP
4404            NOP
4405 3EB3       LD      A,0B3H      Overlay request code for SYS1/SYS
                                    option 30
4407 EF         RST     28H         Load and execute overlay
4408            NOP
4409 C3B044     JP      44B0H       Overlay call to error processor
440C C3B444     JP      44B4H       Make overlay call to load DEBUG
440F 00         NOP
*
*            ----  SYS0 Service Routine Entry Points  ----
*
4410 C39645     JP      4596H       Add address to interrupt task list
4413 C39345     JP      4593H       Remove address from task list
4416 C3A545     JP      45A5H       Replace 1st pointer for current
                                    interrupt task with value in DE
4419 C38E45     JP      458EH       Remove current task from list
*
*              ----  Overlay Load Requests  ----
*
441C 3EC3       LD      A,0C3H      Overlay code for SYS1/SYS
                                    Option 40
441E EF         RST     28H         Load and execute overlay
441F            NOP
4420 3EA4       LD      A,0A4H      INIT overlay request Code
4422 EF         RST     28H         load/execute SYS1/SYS option 20
4423 00         NOP
4424 3E94       LD      A,94H       OPEN overlay request code
4426 EF         RST     28H         Load/execute SYS2/SYS option 10
4427 00         NOP
4428 3E95       LD      A,95H       CLOSE overlay request code
442A EF         RST     28H         load and execute SYS3/SYS option 10
442B 00         NOP
442C 3EA5       LD      A,0A5H      KILL overlay request
442E EF         RST     28H         Load and execute SYS3/SYS option 20
442F 00         NOP
*
*          ----  SYS0 Service Routine Entry Points  ----
*
4430 C3164C     JP      4C16H       Load file specified by DCB in DE
4433 C3064C     JP      4C06H       Load file from system DCB
*
*      ----  Entry Points for Nucleus I/O Operations  ----
*
4436 C36D47     JP      476DH       Read
4439 C38B47     JP      478BH       Write
443C C3A847     JP      47A8H       Verify
443F C35647     JP      4756H       Rewind
4442 C30047     JP      4700H       Position to specified record
4445 C33747     JP      4737H       Backspace, 1 record
4448 C35F47     JP      475FH       Skip to end of file
*
*       ----  SYS0 Service Routine Entry Points  ----
*
4467 C3CF44     JP      44CFH       Send message to video
446A C3DF44     JP      44DFH       Send message to printer
446D C3B74C     JP      4CB7H       Display clock
4470 C3D24C     JP      4CD2H       Display date
```

```
*
*          ---- Overlay Load Requests ----
*
4473 3ED3       LD      A,0D3H          Code for SYS1/SYS option 50
4475 EF         RST     28H             Load and execute overlay
4476 3EE3       LD      A,0E3H          Code for SYS1/SYS option 60
4478 EF         RST     28H             Load and execute overlay
*
*          ----  System DCB (32 bytes)  ----
*
4480            DEFM    'NAMENAME/EXT.PASSWORD:N
*
*          ----  System DCB (16 bytes)  ----
*
44A0            DEFB    80H             Open flag
44A1            DEFB    00              Access flags
44A2            DEFB    00H             Reserved
44A4            DEFW    4D00H           Sector buffer address
44A5            DEFB    00              Offset to next record
44A6            DEFB    00              Drive number
44A7            DEFB    00              Overflow pointer
44A8            DEFB    00              EOF byte address
44A9            DEFB    00H             Record size
44AA            DEFW    0000H           Next record number
44AC            DEFB    FFH             Number of records
44AD            DEFB    FFH
44AE            DEFB    00              Track number
44AF            DEFB    00              Starting number of granules
                                        and number of granules
*
*          ---- Load Error Message Processor ----
*
44B0 F5         PUSH    AF              Save error code
44B1 3E86       LD      A,86H           Overlay request code for SYS4/SYS
44B3 EF         RST     28H             Load and execute error processor
*
*            ---- Load DEBUG ----
*
44B4 F5         PUSH    AF              Save A-register
44B5 3E87       LD      A,87H           Overlay request code for SYS5/SYS
44B7 EF         RST     28H             load and execute DEBUG processor
*
*      ---- Enter Driver. Called From LEVEL II ----
*
44B8 E5         PUSH    HL              Copy DCB address
44B9 DDE1       POP     IX              From HL to IX
44BB DD6E01     LD      L,(IX+01H)      Load driver address LSB
44BE DD6602     LD      H,(IX+02H)      Load driver address MSB
44C1 DD7E00     LD      A,(IX+00H)      Get device code
44C4 FE10       CP      10H             Is this DCB linked to another
44C6 28F0       JR      Z,44B8H         Jump if yes
44C8 D2AE47     JP      NC,47AEH        Jump if disk DCB
44CB 78         LD      A,B             A = op code
44CC FE02       CP      02H             Compare op code to write
44CE E9         JP      (HL)            Enter driver
*
*          ---- Send Message to Video ----
*
44CF E5         PUSH    HL              Save beginning message addess
44D0 7E         LD      A,(HL)          Get a char. from caller's buffer
44D1 FE03       CP      03H             Test for 03 - end of message.
44D3 2808       JR      Z,44DDH         Exit if end of message.
44D5 CD3300     CALL    0033H           Display on video
44D8 23         INC     HL              Bump to next display char.
44D9 FE0D       CP      0DH             Test for CR - end of message
44DB 20F3       JR      NZ,44D0H        Jump if not CR
44DD E1         POP     HL              End of message encountered
44DE C9         RET                     Restore caller's HL & return
*
*          ---- Send Message to Printer ----
*
44DF E5         PUSH    HL              Save origin of message
44E0 7E         LD      A,(HL)          Fetch a byte of message
44E1 FE03       CP      03H             End of message
44E3 2808       JR      Z,44EDH         Yes, exit
44E5 CD3B00     CALL    003BH           No, print character
44E8 23         INC     HL              Bump to next character
44E9 FE0D       CP      0DH             Was last char. a CR
44EB 20F3       JR      NZ,44E0H        No, get & print char.
44ED E1         POP     HL              Yes, restore start of msg. addr.
44EE C9         RET                     and return to caller
44EF 00         NOP
*
*          ---- Interrupt Task List Address ----
*
* This list contains pointers to addresses of tasks to be executed
* on every clock interrupt. The list may be divided into two parts.
* Entries from 4510-4516 are executed unconditionally on every
* clock interrupt (every 25 milliseconds). Entries from 4500-450E
* are executed once every 8 interrupts (200 milliseconds). Task
* execution is controlled by the subroutine at 4560 which is
* called by the interrupt processor. Location 4040 contains a
* counter used to select the next task to be executed from the
* first part of the list.
*
4500            DEFW    45A3H           Address of task 0
4502            DEFW    45A3H           Address of task 1
4504            DEFW    45A3H           Address of task 2
4506            DEFW    45A3H           Address of task 3
4508            DEFW    45A3H           Address of task 4
450A            DEFW    45A3H           Address of task 5
450C            DEFW    45A3H           Address of task 6
450E            DEFW    45A3H           Address of task 7
4510            DEFW    45A3H           Address of task 8
4512            DEFW    45A3H           Address of task 9
4514            DEFW    45A3H           Address of task 10
4516            DEFW    45A3H           Address of task 11
*
*          ---- Interrupt Service Routine ----
*          Entered for all Maskable Interrupts
*
4518 E5         PUSH    HL              Save interrupt
4519 F5         PUSH    AF              Context
451A 3AE037     LD      A,(37E0H)       Get interrupt status byte
451D 214B40     LD      HL,404BH        Addr. to save interrupt status
4520 77         LD      (HL),A          Save interrupt status byte
4521 2C         INC     L               HL = 404C (addr. of interrupt
                                        service routine mask byte)
4522 A6         AND     (HL)            Compare mask to interrupt status
4523 2808       JR      Z,452DH         Jump if no match, ignore interrupt
4525 2C         INC     L               Bump HL to 404D, start of
                                        interrupt processor's addr. list
4526 1F         RRA                     Test a bit in the interrupt
                                        status. Start at Bit 7 and
                                        work towards Bit 0
4527 380F       JR      C,4538H         Jump if ON bit found
4529 2C         INC     L               Bump HL to address of next
                                        interrupt processor, must
                                        bump twice, second one at 4525
452A B7         OR      A               Any remaining interrupt bits?
452B 20F8       JR      NZ,4525H        If so, go test for specific bit
452D 3A4038     LD      A,(3840H)       If not, load BREAK key row
4530 E604       AND     04H             Isolate BREAK bit
4532 201B       JR      NZ,454FH        Jump if BREAK active. Go test
                                        if DEBUG should be loaded.
4534 F1         POP     AF              Exit interrupt service routine
4535 E1         POP     HL              Restore interrupted context
4536 FB         EI                      Enable interrupts
4537 C9         RET                     Return to point of interrupt
*
*   --- Call Interrupt Task Routine Associated With On List ---
*
4538 F5         PUSH    AF              Save interrupt service
                                        routine context
4539 C5         PUSH    BC              Save interrupt context
453A D5         PUSH    DE              Save interrupt context
453B E5         PUSH    HL              Save interrupt context
453C DDE5       PUSH    IX              Saving done
453E 114745     LD      DE,4547H        Return address after executing
                                        interrupt processor routine
4541 D5         PUSH    DE              To stack (simulate a CALL)
4542 5E         LD      E,(HL)          Load LSB of processor address
4543 2C         INC     L               Bump to MSB of address in list
4544 56         LD      D,(HL)          Load MSB of processor address
4545 EB         EX      DE,HL           Interrupt processor addr. to HL
4546 E9         JP      (HL)            Jump to interrupt processor
*
*     ---- Return Point for Interrupt Task Routines ----
*
4547 DDE1       POP     IX              Restore interrupted
                                        service context
4549 E1         POP     HL              Restore context
454A D1         POP     DE              Restore context
454B C1         POP     BC              Restore context
454C F1         POP     AF              Restoring done
454D 18DA       JR      4529H           Loop. Look for more bits
                                        in status byte
*
* Active BREAK key detected during interrupt processing. Test if
* DEBUG is active. If not, return to point of interrupt, else clear
* active flag, load and execute DEBUG.
*
454F 211543     LD      HL,4315H        HL = address of DEBUG
                                        active flag
4552 AF         XOR     A               Load comparison value for
                                        DEBUG inactive
4553 BE         CP      (HL)            Is DEBUG active
4554 28DE       JR      Z,4534H         Jump if no
4556 77         LD      (HL),A          Yes, reset flag to inactive
4557 2C         INC     L               Bump to LSB of addr. containing
                                        DEBUG load request
4558 7E         LD      A,(HL)          Load LSB
4559 2C         INC     L               Bump to MSB of addr. containing
                                        DEBUG load request
455A 66         LD      H,(HL)          Load MSB
455B 6F         LD      L,A             Form 400F in HL. Address of
                                        instrucion to load DEBUG
455C F1         POP     AF              Restore interrupted context
455D E3         EX      (SP),HL         Restore interrupted HL
                                        Pushs 400F onto stack
455E FB         EI                      Allow interrupts
455F C9         RET                     Go load and execute DEBUG
*
*          ---- Clock Interrupt Routine ----
*
4560 3E08       LD      A,08H           Index for 8th entry in interrupt
                                        task list table
4562 CD7B45     CALL    457BH           Go execute task
4565 3E09       LD      A,09H           Index for 9th entry in interrupt
                                        task list table
4567 CD7B45     CALL    457BH           Go execute task
456A 3E0A       LD      A,0AH           Index for 10th entry in interrupt
                                        task list table
456C CD7B45     CALL    457BH           Go execute task
456F 3E0B       LD      A,0BH           Index for 11th entry in interrupt
                                        task list table
4571 CD7B45     CALL    457BH           Go execute task
4574 214040     LD      HL,4040H        Index address for tasks to be
                                        executed every 200 milliseconds
4577 34         INC     (HL)            Bump index
4578 7E         LD      A,(HL)          Fetch index
4579 E607       AND     07H             Restrict index to range 0-7,
                                        then fall into subroutine
                                        that locates task address
                                        and executes task
457B 07         RLCA                    Index times two gives LSB of
                                        interrupt task list address
```

```
457C 6F          LD      L,A             Index times two to L reg. so
457D 2645        LD      H,45H           Interrupt task list address can
                                         be formed in HL-register
457F 22A645      LD      (45A6H),HL      Save current task list address
*
*       Load address for 1st level of indirection into DE
*
4582 5E          LD      E,(HL)          Fetch LSB of address for 1st level
4583 2C          INC     L               Bump to MSB of address
4584 56          LD      D,(HL)          Fetch MSB of address for 1st level
4585 D5          PUSH    DE              Address for 1st level of
                                         indirection to stack
4586 DDE1        POP     IX              as a convenient way to load IX
4588 EB          EX      DE,HL           Addr. for 1st level of
                                         indirection to HL
*
*       Load address for 2nd level of indirection
*
4589 5E          LD      E,(HL)          Fetch LSB of 2nd level
458A 23          INC     HL              Bump to MSB of address
458B 56          LD      D,(HL)          Fetch MSB of address for 2nd
                                         level of indirection
458C EB          EX      DE,HL           2nd level (task address) to HL
458D E9          JP      (HL)            Go execute task. Return to caller
                                         of 457B or 4560
*
*         Replace address for 1st pointer in
*         interrupt table with value in DE
*
458E D1          POP     DE              Clear stack
458F 3AA645      LD      A,(45A6H)       Fetch LSB of 1st pointer for last
                                         interrupt task executed
4592 0F          RRCA                    Divide by 2 to get index into
                                         interrupt task list
4593 11A345      LD      DE,45A3H        New 1st pointer address if 45A6
                                         contains an address from
                                         interrupt task list
4596 FE0C        CP      0CH             Compare index for last task to
                                         execute. (maximum of 12)
4598 D0          RET     NC              Return if index not past end of
                                         interrupt task list
4599 07          RLCA                    Reform address using Index
459A 6F          LD      L,A             And form
459B 2645        LD      H,45H           Address of interrupt task
                                         list in HL
459D F3          DI                      Kill interrupts while we adjust
                                         interrupt task list table
459E 73          LD      (HL),E          Store 45A3 (1st pointer of
                                         last index)
459F 2C          INC     L               This will cause all
45A0 72          LD      (HL),D          Subsequent calls to this index
45A1 FB          EI                      to be ignored since 45A3 points
                                         to 45A2 which contains a RET
45A2 C9          RET                     Return to caller
*
*
*
45A3 A245        DEFW    45A2H
45A5 210000      LD      HL,0000         Addr. of current interrupt task
45A8 5E          LD      E,(HL)          E = LSB of task list address
45A9 23          INC     HL              Bump to MSB of address
45AA 56          LD      D,(HL)          D = MSB of task list address
45AB EB          EX      DE,HL           Move task list addr. to HL so new
45AC D1          POP     DE              1st pointer can be stored
                                         clear stack
45AD 18EE        JR      459DH           Store address of first pointer
                                         for current interrupt task
*
*           ----  Clock Maintenance Routine  ----
*            Called from Interrupt Service Routine
*                  if CLOCK (ON)
*
45AF B245        DEFW    45B2H
45B1 05          DEFB    5               Count of times entered
45B2 DD3502      DEC     (IX+02H)        Have we been entered 5 times?
                                         (has 1 second elapsed)
45B5 C0          RET     NZ              Ret to interrupt processor if not
45B6 DD360205    LD      (IX+02H),05H    Yes. Reset elapsed time counter
45BA 0603        LD      B,03H           Maximum number of values to update
45BC 214140      LD      HL,4041H        Starting address for time (seconds)
45BF 11EF45      LD      DE,45EFH        Starting address for maximum
                                         values for each time period
45C2 34          INC     (HL)            Bump a time period, seconds,
                                         then minutes, then hours
45C3 1A          LD      A,(DE)          Fetch maximum value for period
                                         being incremented
45C4 96          SUB     (HL)            Compare period to max. unit value
45C5 C0          RET     NZ              Return if unit not max. value
45C6 77          LD      (HL),A          Else reset period to zero
45C7 2C          INC     L               And bump increment and
45C8 1C          INC     E               Test addresses to next unit
45C9 10F7        DJNZ    45C2H           Loop till periods incremented
                                         or no overflow from one
                                         period to the next
*
*                   ----  Update DATE  ----
*                  if 24 hours have passed
*
45CB 2C          INC     L               Skip over year value for date
45CC 34          INC     (HL)            Bump day value for date
45CD 2C          INC     L               Skip to month value for date
45CE 7E          LD      A,(HL)          Fetch month value
45CF 2D          DEC     L               Backspace to day value
45D0 3D          DEC     A               Decrement month
45D1 83          ADD     A,E             So it can be used as an index
                                         into month table
45D2 5F          LD      E,A             Form month table address
45D3 1A          LD      A,(DE)          Fetch days for current month
45D4 BE          CP      (HL)            Compare with current days
45D5 D0          RET     NC              Exit if month has not rolled over
45D6 7E          LD      A,(HL)          Refetch current day number
45D7 FE1E        CP      1EH             Compare with 30
45D9 3006        JR      NC,45E1H        Jump if day is 30 or 31
                                         else, it's FEB
45DB 2D          DEC     L               Backspace to year value for date
45DC 7E          LD      A,(HL)          Fetch year
45DD 2C          INC     L               Skip forward to day
45DE E603        AND     03H             Test for leap year
45E0 C8          RET     Z               Exit if not leap year
45E1 3601        LD      (HL),01H        Month rolled over. Reset day to 1
45E3 2C          INC     L               Bump to month
45E4 34          INC     (HL)            Increment month value
45E5 7E          LD      A,(HL)          Fetch month value
45E6 D60D        SUB     0DH             Test for year rollover
45E8 D8          RET     C               Exit if not end of year
45E9 3601        LD      (HL),01H        Reset month to 1
45EB 2D          DEC     L               Backspace over month
45EC 2D          DEC     L               And day
45ED 34          INC     (HL)            Then increment year
45EE C9          RET                     Return to interrupt processor
*
*              ----  Maximum Values for Clock Periods  ----
*
45EF 3C          DEFB    3CH             Decimal 60 seconds
45F0 3C          DEFB    3CH             Decimal 60 minutes
45F1 18          DEFB    18H             Decimal 24 hours
*
*
*                     ----  Month Table  ----
*               Contains Number of Days in Each Month.
*
*
45F2             DEFB    1FH     JAN 31
45F3             DEFB    1CH     FEB 28
45F4             DEFB    1FH     MAR 31
45F5             DEFB    1EH     APR 30
45F6             DEFB    1FH     MAY 31
45F7             DEFB    1EH     JUN 30
45F8             DEFB    1EH     JUL 31
45F9             DEFB    1EH     AUG 31
45FA             DEFB    1EH     SEP 30
45FB             DEFB    1EH     OCT 31
45FC             DEFB    1FH     NOV 30
45FD             DEFB    1FH     DEC 31
*
*           ----  Select Drive Specified in C-Register  ----
*
4600 E5          PUSH    HL              Save caller's HL
4601 210843      LD      HL,4308H        Address of last drive selected
4604 7E          LD      A,(HL)          Load last drive selected
4605 B9          CP      C               Compare with drive requested
4606 2004        JR      NZ,460CH        Jump if different drive. Must
                                         construct new drive select mask
4608 2C          INC     L               Bump to last drive select mask
4609 7E          LD      A,(HL)          Load drive select mask
460A 1818        JR      4624H           Go select drive
460C 71          LD      (HL),C          Update last drive selected
460D 2643        LD      H,43H           Form origin of last track Table
460F 6F          LD      L,A             Form origin of current drive
4610 CD9843      CALL    4398H           Wait for controller, not busy
4613 3AED37      LD      A,(37EDH)       Load current track,
4616 77          LD      (HL),A          save in last track selected table
4617 79          LD      A,C             A = drive number
4618 6F          LD      L,A             Reform last track addr. for drive
4619 7E          LD      A,(HL)          Fetch last track selected this drive
461A 32ED37      LD      (37EDH),A       Update track register to
                                         current track number
461D 79          LD      A,C             A = drive number
461E CD3946      CALL    4639H           Construct drive select mask
4621 320943      LD      (4309H),A       And save for future use
4624 6F          LD      L,A             Temporary storage while A in use
4625 3AEC37      LD      A,(37ECH)       Fetch controller status
4628 07          RLCA                    Shift NOT READY bit into CARRY
4629 7D          LD      A,L             Restore drive select mask
462A 32E137      LD      (37E1H),A       Select drive
462D 3008        JR      NC,4637H        Jump if drive ready
462F 210000      LD      HL,0000H        Drive not ready. Delay one sec.
4632 2B          DEC     HL              Decrement 16 bit counter
4633 7C          LD      A,H             Combine MSB
4634 B5          OR      L               And LSB of counter
4635 20FB        JR      NZ,4632H        Loop if counter not zero
4637 E1          POP     HL              Restore caller's HL
4638 C9          RET                     Return to caller
*
*   -- Construct Unit Select Mask for Value in A-Register --
*
4639 E607        AND     07H             Restrict drive to 0-3
463B 07          RLCA                    Shift drive number left
463C 07          RLCA                    3 bit positions
463D 07          RLCA                    to combine with SET op code
463E F6C7        OR      0C7H            Synthesize SET (A),A instruction
4640 324546      LD      (4645H),A       Save synthesized instruction
4643 AF          XOR     A               Zero A
4644 CBC7        SET     00H,A           Turn on drive number bit
4646 C9          RET                     Return to caller
*
*    ----  Select Drive and Issue Seek if not on Track  ----
*
4647 CD0046      CALL    4600H           Select drive
464A CD6946      CALL    4669H           Wait for controller not busy
464D 3AED37      LD      A,(37EDH)       Fetch last track selected
4650 BA          CP      D               And compare with required track
4651 ED53EE37    LD      (37EEH),DE      Update track/sector registers
4655 C8          RET     Z               Ret if we're on the right track
4656 3E1F        LD      A,1FH           Else load SEEK op code
4658 F5          PUSH    AF              Save op code on stack
4659 CD6946      CALL    4669H           Wait for controller not busy
465C F1          POP     AF              Restore SEEK op code
465D 32EC37      LD      (37ECH),A       And send to controller
4660 C9          RET                     Return to caller
```

```
*
*           Send Command in A-register to Disk Cantroller
*                Returns When Command Processed
*                    (Controller Not Busy)
*
4661 CD5846     CALL    4658H       Send command to controller
4664 F5         PUSH    AF          Delay for 19 microseconds
4665 F1         POP     AF          Delay
4666 00         NOP                 Delay
4667 00         NOP                 Delay
4668 00         NOP                 Delay
4669 CDAC43     CALL    43ACH       Select drive. Get status
466C CB47       BIT     00H,A       Test if controller busy
466E C8         RET     Z           Exit if controller not busy
466F 18F8       JR      4669H       Loop till controller not busy
*
*                 ---- Disk Driver ----
*
*       Calling Sequence:
*
*             LD    A,CODE    A = controller op code for
*                                   read or write
*             CALL  4671      C = Unit. DE = Track/Sector
*             DEFB  0         error retry count
*             DEFB  0         bias value Added to first
*                             'on' list in 2 station
*             LD    A, (BC)       use for a WRITE command
*             LD    (DE), A       use for a WRITE command
*                        ----OR----
*             LD    A, (DE)       use for a READ command
*             LD    (BC), A       use for a READ command
*
4671 329346     LD      (4693H),A   Save disk controller op code
4674 E3         EX      (SP),HL     Parameter list address to HL
                                    Buffer address to stack
4675 C5         PUSH    BC          Save caller's BC
4676 46         LD      B,(HL)      Fetch retry count
4677 23         INC     HL          Bump to next parameter
4678 7E         LD      A,(HL)      Get bias count to add to first
                                    'ON' bit in status if error
4679 23         INC     HL          Bump to next parameter
467A 32D346     LD      (46D3H),A   Save error bias value
467D 7E         LD      A,(HL)      Get FETCH/STORE instruction
                                    to or from controller
467E 23         INC     HL          Bump to next parameter
467F 66         LD      H,(HL)      Get FETCH/STORE instruction
                                    to or from data buffer
4680 6F         LD      L,A         Assemble controller/buffer
                                    reference commands
4681 22A746     LD      (46A7H),HL  And save in main driver loop
4684 E1         POP     HL          Caller's BC to HL
4685 E3         EX      (SP),HL     HL = buffer address
4686 CD4746     CALL    4647H       Go select unit. Position to
                                    track and sector in DE
4689 CD6946     CALL    4669H       Wait for controller to do it
468C C5         PUSH    BC          Save retry count/unit number
468D D5         PUSH    DE          Save track/sector number
468E E5         PUSH    HL          Save buffer address
468F 21EC37     LD      HL,37ECH    Controller command/status register
4692 3600       LD      (HL),00H    Issue READ/WRITE command
4694 11EF37     LD      DE,37EFH    Addr of controller data register
4697 C1         POP     BC          Delaly and buffer addr. to BC
4698 C5         PUSH    BC          Controller needs time to see command
4699 C1         POP     BC          before requesting status, otherwise
469A C5         PUSH    BC          Status could overwrite command
469B F3         DI                  Kill interrupts because of PIO code
469C 1803       JR      46A1H       Go get status, begin transfer
469E 0F         RRCA                Controller busy bit to CARRY
469F 300C       JR      NC,46ADH    Go if not busy (Op complete)
46A1 7E         LD      A,(HL)      Fetch controller status
46A2 CB4F       BIT     01H,A       Ready to transfer a data byte?
46A4 28F8       JR      Z,469EH     Jump if no, assume op complete
46A6 F3         DI                  unnecessary instruction
46A7 00         NOP                 FETCH/STORE byte from/to controller
46A8 00         NOP                 FETCH/STORE byte from/to buffer
46A9 03         INC     BC          Bump buffer addr to next byte
46AA C3A146     JP      46A1H       Loop till controller not busy
*
*                 ---- Operation Complete ----
*                      Check Status and
*                      Test for Errors
*
46AD FB         EI                  Transfer done. Enable interrupts
46AE 7E         LD      A,(HL)      Get controller status
46AF E67C       AND     7CH         Strip NOT READY bit off
46B1 36D0       LD      (HL),0D0H   Force interrupt to clear controller
46B3 E1         POP     HL          Restore buffer address
46B4 D1         POP     DE          Restore track/sector address
46B5 C1         POP     BC          Restore retry count/unit number
46B6 2823       JR      Z,46DBH     Jump if no errors
46B8 CB57       BIT     02H,A       Test for LOST DATA status
46BA 2013       JR      NZ,46CFH    Jump if LOST DATA
46BC FE20       CP      20H         Test for WRITE FAULT
46BE 2811       JR      Z,46D1H     Jump if WRITE FAULT
46C0 F5         PUSH    AF          Save error status
46C1 3EFF       LD      A,0FFH      Force unit to be selected (modify
46C3 320843     LD      (4308H),A   last unit selected byte)
46C6 CD0046     CALL    4600H       Select unit
46C9 3E0B       LD      A,0BH       Restore head command
46CB CD5846     CALL    4658H       Issue restore command
46CE F1         POP     AF          Restore last disk status
46CF 10B5       DJNZ    4686H       Loop for 'retry' counts
46D1 47         LD      B,A         Status for last operation
46D2 3E00       LD      A,00H       Bias value to add to error number
46D4 CB08       RRC     B           Right shift error status
46D6 3803       JR      C,46DBH     until first 'ON' bit detected
46D8 3C         INC     A           For each bit shifted, add 1 to bias
46D9 18F9       JR      46D4H       Loop till an error bit is in CARRY
46DB C1         POP     BC          No error, restore retry count/unit
46DC C9         RET                 Return to caller
*
*    ----  Read Subroutine Used for Calling Disk Driver  ----
*          Driver Returns to Caller of Read Subroutine
*
46DD 3E88       LD      A,88H       Read command
46DF CD7146     CALL    4671H       Call disk drives
46E2 0A         DEFB    0AH         Retry count if error
46E3 01         DEFB    01H         Bias to be added to the number of
                                    the first 'ON' bit if error
46E4 1A         LD      A,(DE)      Fetch data byte from controller
46E5 02         LD      (BC),A      Data byte to caller's buffer
*
*   ----  Write Subroutine Used for Calling Disk Driver  ----
*         Driver Returns to Caller of Write Subroutine
*
46E6 3EA8       LD      A,0A8H      WRITE command. Generate 'FB'
                                    data address mark
46E8 CD7146     CALL    4671H       Call disk driver
46EB 05         DEFB    05H         Retry count if error
46EC 09         DEFB    09H         Bias added to number of first 'ON'
                                    bit in status if error
46ED 0A         LD      A,(BC)      Fetch data byte from data buffer
46EE 12         LD      (DE),A      Send data byte to controller
*
*   ----  Write Subroutine Used for Calling Disk Driver  ----
*         Driver Returns to Caller of Write Subroutine
*
46EF 3EA9       LD      A,0A9H      WRITE command. Generate 'FA'
                                    data address mark
46F1 18F5       JR      46E8H       Use normal write disk call
*
*    ----  Read Subroutine Used for Calling Disk Driver  ----
*          Driver Returns to Caller of Read Subroutine
*
46F3 3E88       LD      A,88H       READ command
46F5 CD7146     CALL    4671H       Call disk driver
46F8 05         DEFB    05H         Retry count if error
46F9 01         DEFB    01H         Status bias (see comment at 46EC)
46FA 1A         LD      A,(DE)      Fetch data byte from controller
46FB 00         NOP                 Do not store data in buffer
*
*      ----  Position File to Specified Record Number  ----
*
4700 CD9248     CALL    4892H       Set IX to DCB addr. Save register
                                    Insure file OPEN
4703 DDCB01F6   SET     06H,(IX+01H) Signal position call
4707 DDCB017E   BIT     07H,(IX+01H) Test for physical or logical I/O
470B 280A       JR      Z,4717H     Jump if physical I/O
470D 60         LD      H,B         Move record number from BC
470E 69         LD      L,C         to HL, so it can be divided
470F DD7E09     LD      A,(IX+09H)  Fetch record size
4712 CD6A4B     CALL    4B6AH       Divide record no. by record size
4715 44         LD      B,H         Move quotient (relative physical
                                    record number) to BC
4717 DD7705     LD      (IX+05H),A  Save record offset within sector
471A C5         PUSH    BC          Save relative physical record no.
471B CD7848     CALL    4878H       Flush sector buffer to disk
471E C1         POP     BC          Restore relative physical record no.
471F C0         RET     NZ          Early exit if any error when
                                    writing sector buffer
4720 DD710A     LD      (IX+0AH),C  Move relative sector
4723 DD700B     LD      (IX+0BH),B  To DCB as next record number
4726 DDCB01EE   SET     05H,(IX+01H) Force read on next file access
472A CDB948     CALL    48B9H       Test, for end of file
472D C0         RET     NZ          Early exit if end of file
472E CDDC48     CALL    48DCH       Get track/sector for sector no.
4731 C0         RET     NZ          Go if error while translating record
                                    number to disk address
4732 CD4746     CALL    4647H       Select drive and find track
4735 AF         XOR     A           Signal good status
4736 C9         RET                 Return to caller
*
*              ----  Backspace Operation  ----
*
4737 CD9248     CALL    4892H       Set up exit sequence
473A DD4E0A     LD      C,(IX+0AH)  Load BC with
473D DD460B     LD      B,(IX+0BH)  next sector number
4740 DDCB017E   BIT     07H,(IX+01H) Test for logical vs phy I/O
4744 280D       JR      Z,4753H     Jump if phy I/O, else
4746 DD7E09     LD      A,(IX+09H)  Get record size from DCB
4749 B7         OR      A           Test if record size 256 bytes
474A 2807       JR      Z,4753H     Jump if it is
474C ED44       NEG                 Else negate record size and
474E DD8605     ADD     A,(IX+05H)  Subtract record length from
                                    position in sector buffer
4751 38C4       JR      C,4717H     Go if record crosses sector line
4753 0B         DEC     BC          Decrement sector no. if physical I/O
                                    or record doesn't cross sector line
4754 18C1       JR      4717H       Update DCB, Flush buffer & return
*
*                  ----  Rewind File  ----
*           Use POSN Code to Position to Record 0
*
4756 CD9248     CALL    4892H       Save reg. Set up exit sequence
4759 010000     LD      BC,0000H    Record number for POSN call
475C AF         XOR     A           Zero offset in sector buffer
475D 18B8       JR      4717H       Use POSN code to rewind file
*
*                  ----  Skip to end of File  ----
*
475F CD9248     CALL    4892H       Save reg. set up exit sequence
4762 DD4E0C     LD      C,(IX+0CH)  Load number of records
4765 DD460D     LD      B,(IX+0DH)  in file into BC
4768 CDD143     CALL    43D1H       A = offset to EOF in last sector
476B 18AA       JR      4717H       Use POSN code to update DCB
```

```
*
*                ----  Read Processing  ----
*
476D CD9248     CALL    4892H           Save reg. Set up exit sequence
4770 CD7848     CALL    4878H           Purge buffer if necessary
4773 C0         RET     NZ              Exit if any error while purging
4774 DDCB017E   BIT     07H,(IX+01H)    Test for phy vs logical I/O
4778 CA0E48     JP      Z,480EH         Jump if we must read a sector
                                        Buffer not ready, or sector I/O
*
*                  ----  Logical READ  ----
*
* Transfer record from sector buffer to user's record area byte
* by byte. The subroutine at 47BB returns the next byte from
* the sector buffer. If the buffer is exhaused, the next sector
* will be read.
*
477B DD4609     LD      B,(IX+09H)      B = record size
477E E5         PUSH    HL              Save user's record area address
477F C5         PUSH    BC              Save byte count (record size)
4780 CDBB47     CALL    47BBH           Get a byte from sector buffer
4783 C1         POP     BC              Restore byte count for transfer
4784 E1         POP     HL              Restore user rec. area address
4785 C0         RET     NZ              Exit if any error during read
4786 77         LD      (HL),A          Byte to caller record area
4787 23         INC     HL              Bump caller's buffer addr.
4788 10F4       DJNZ    477EH           Loop till (B) bytes moved
478A C9         RET                     Return when record moved
*
*          ----  Beginning of WRITE Processing  ----
*
478B CD9248     CALL    4892H           Save reg. Set up exit sequence
478E 325E48     LD      (485EH),A       Initialize VERIFY flag
4791 DDCB017E   BIT     07H,(IX+01H)    Test for logical vs physical I/O
4795 CA3E48     JP      Z,483EH         Jump if physical I/O
*
*                  ----  Logical WRITE  ----
*
* Transfer record from user's record area to the file sector
* buffer byte by byte. The subroutine at 47DA stores the next byte
* in the sector buffer. If the buffer is full, it is written
* to disk.
*
4798 DD4609     LD      B,(IX+09H)      Get record length in bytes
479B 7E         LD      A,(HL)          Fetch one byte of record from
                                        user's record area
479C 23         INC     HL              Bump to next data byte
479D E5         PUSH    HL              Save fetch address
479E C5         PUSH    BC              Save count of characters moved
479F CDDA47     CALL    47DAH           Store char. in sector buffer
47A2 C1         POP     BC              Restore count of chars. moved
47A3 E1         POP     HL              And current user record address
47A4 C0         RET     NZ              Exit if any error during write
47A5 10F4       DJNZ    479BH           Loop till entire record moved
47A7 C9         RET                     Return to caller
*
*                  ----  VERIFY Processing  ----
*
47A8 CD9248     CALL    4892H           Save reg. set up exit sequence
47AB 3C         INC     A               Load a 1 to specify VERIFY
47AC 18E0       JR      478EH           Then use WRITE subroutine
*
*          ----  General Purpose Driver Entry  ----
*             Called From Driver Entry at 44BB
*
47AE CD9248     CALL    4892H           Save reg. set up exit sequence
47B1 DDCB01FE   SET     07H,(IX+01H)    Signal logical I/O
47B5 78         LD      A,B             Fetch operation code
47B6 FE02       CP      02H             Test for read or write
47B8 79         LD      A,C             Save data for write op code
47B9 301F       JR      NC,47DAH        Jump if write operation
*
*                  ----  Read Operation  ----
*
47BB CDB948     CALL    48B9H           Tests for end of file
47BE C0         RET     NZ              Return if EOF. A = 1D/1C
47BF DDCB016E   BIT     05H,(IX+01H)    Need another sector for buffer?
47C3 2804       JR      Z,47C9H         No, get next byte from record
47C5 CD0E48     CALL    480EH           Read next sector from file
47C8 C0         RET     NZ              Exit if EOF or other error
47C9 CD8348     CALL    4883H           Put addr of next record in DE
47CC AF         XOR     A               Clear status
47CF 1A         LD      A,(DE)          Fetch character from sector buffer
47CE F5         PUSH    AF              Save character fetched
47CF DD3405     INC     (IX+05H)        Bump to next byte in sector buffer
47D2 2004       JR      NZ,47D8H        Jump if buffer not empty
47D4 DDCB01EE   SET     05H,(IX+01H)    Signal read required
47D8 F1         POP     AF              Restore character fetched
47D9 C9         RET                     And return to caller
*
*                  ---- Write Operation----
*
47DA F5         PUSH    AF              Save character to be written
47DB DDCB016E   BIT     05H,(IX+01H)    Test if sector buffer full
47DF 2808       JR      Z,47E9H         Jump if room in sector buffer
47E1 CD2D48     CALL    482DH           Read next sector from file
47E4 2803       JR      Z,47E9H         Jump if no error during read
47E6 E3         EX      (SP),HL         Error, clear character from stack
47E7 E1         POP     HL              Clear stack
47E8 C9         RET                     And return to caller
47E9 CD8348     CALL    4883H           Put addr of next byte position
                                        in sector buffer in DE
47EC F1         POP     AF              Get character to be written
47ED 12         LD      (DE),A          Move it to sector buffer
47EE 13         INC     DE              Bump to next addr. in buffer
47EF DDCB01E6   SET     04H,(IX+01H)    Flag buffer as updated
                                        Must be purged
47F3 DD3405     INC     (IX+05H)        Bump to next position in buffer
47F6 CDB948     CALL    48B9H           Test for EOF access permission
47F9 2006       JR      NZ,4801H        Jump if access error
47FB DDCB0176   BIT     06H,(IX+01H)    Record number to be incremented
47FF 2003       JR      NZ,4804H        Jump if not
4801 DD7108     LD      (IX+08H),C      Update EOF addr in sector buffer
4802 DD750C     LD      (IX+0CH),L      Update number of records in file
4804 79         LD      A,C             Current sector buffer offset
4805 B7         OR      A               Set status flags for offset
4806 2023       JR      NZ,482BH        Go if end of buffer not reached
4808 DDCB01EE   SET     05H,(IX+01H)    Buffer full, force read
                                        on next access
480C 1830       JR      483EH           Go to common exit for writes
*
*          ----  Read Next Sector From File  ----
*
480E DD7E01     LD      A,(IX+01H)      Fetch access level
4811 E607       AND     07H             Isolate permission flags
4813 FE06       CP      06H             Test for READ/WRITE access
4815 3804       JR      C,481BH         Jump if no read protection
4817 3E25       LD      A,25H           Error code: illegal access
                                        attempted on protected file
4819 B7         OR      A               Set status flags for error code
481A C9         RET                     Return to caller
481B CDB948     CALL    48B9H           Test for end of file
481E C0         RET     NZ              Return if end of file
481F CD2D48     CALL    482DH           Read next sector from file
4822 C0         RET     NZ              Exit if error during read
4823 DD340A     INC     (IX+0AH)        Bump LSB of next record number
4826 2003       JR      NZ,482BH        Jump if no overflow
4828 DD340B     INC     (IX+0BH)        Bump MSB of next record number
482B AF         XOR     A               Signal good status. No error
482C C9         RET                     Return to caller
*
*
*
482D CDDC48     CALL    48DCH           Get disk addr. for next sector
4830 C0         RET     NZ              Early return if any error
4831 DDCB01AE   RES     05H,(IX+01H)    Clear force read next sector flag
4835 DD6E03     LD      L,(IX+03H)      Load HL with
4838 DD6604     LD      H,(IX+04H)      Sector buffer address
483B C3DD46     JP      46DDH           Go read sector and ret to caller
*
*          ----  Continuation of WRITE Processing  ----
*
483E DD7E01     LD      A,(IX+01H)      Fetch access flags
4841 E607       AND     07H             Isolate permission
4843 FE05       CP      05H             Test for EXECUTE only or
                                        restricted file
4845 3804       JR      C,484BH         Jump if neither true, or
4847 3E25       LD      A,25H           ret error for illegal access
4849 B7         OR      A               Set status flags
484A C9         RET                     and return to caller
484B CDDC48     CALL    48DCH           Get disk address for next sector
484E C0         RET     NZ              Exit early if any error while
                                        computing disk address
484F DD6E03     LD      L,(IX+03H)      LSB of sector buffer address
4852 DD6604     LD      H,(IX+04H)      MSB of sector buffer address
4855 CDE646     CALL    46E6H           Write current sector buffer
4858 C0         RET     NZ              Exit if any error during Write
4859 DDCB01A6   RES     04H,(IX+01H)    Signal sector buffer available
485D 3E00       LD      A,00H           Load VERIFY flag
485F B7         OR      A               Set status for VERIFY
4860 C4F346     CALL    NZ,46F3H        Read sector back if VERIFY (ON)
4863 C0         RET     NZ              Exit if any error during read
4864 CDB948     CALL    48B9H           Test for end of file
4867 2006       JR      NZ,486FH        Jump if end of file encountered
4869 DDCB0176   BIT     06H,(IX+01H)    Test if position call
486D 20B4       JR      NZ,4823H        Jump if it is
486F 23         INC     HL              Bump record number
4870 DD750C     LD      (IX+0CH),L      and update number of
4873 DD740D     LD      (IX+0DH),H      records in DCB
4876 18AB       JR      4823H           Update record number and exit
*
*              ----  Purge Sector Buffer  ----
*
4878 DD7E01     LD      A,(IX+01H)      Fetch access flags
487B E690       AND     90H             Isolate logical I/O, write access
487D FE90       CP      90H             Test for logical I/O, write access
487F 28BD       JR      Z,483EH         Go if logical file. test for execute
4881 AF         XOR     A               Signal good status if physical I/O
4882 C9         RET                     Return to caller
*
* Get addr next record in sector buffer in DE. Compute addr
* for next byte to be stored/fetched in current logical record
*
4883 DD7E05     LD      A,(IX+05H)      Current byte offset in sector buffer
4886 4F         LD      C,A             Unnecessary instruction
4887 DD8603     ADD     A,(IX+03H)      Add LSB of buffer addr to offset
488A 5F         LD      E,A             E = LSB of physical buffer addr
488B DD7E04     LD      A,(IX+04H)      Fetch MSB of sector buffer addr
488E CE00       ADC     A,00H           Add any carry from LSB addition
4890 57         LD      D,A             DE = position of record in buffer
4891 C9         RET                     Return to caller
*
*                  ----  Save Registers  ----
*
*      Stack configuration:
*                  Return address    - For original I/O call
*                  IX
*                  HL
*                  DE
*                  BC
*                  48B3              - Restores preceding context
*
4892 1A         LD      A,(DE)          A = First byte of DCB
4893 07         RLCA                    Position OPEN bit
4894 3018       JR      NC,48AEH        Jump if file not open
4896 F1         POP     AF              Rtn addr to AF. Need to put
4897 E3         EX      (SP),HL         Buffer addr in it's place
4898 220C43     LD      (430CH),HL      Buffer addr to 430C
489B ED530A43   LD      (430AH),DE      Save DCB addr
```

```
489F E3         EX      (SP),HL     Restore buffer addr to HL,
                                    return address to stack
48A0 D5         PUSH    DE          Save DCB address on stack
48A1 DDE3       EX      (SP),IX     Save original IX (DCB address)
48A3 E5         PUSH    HL          Caller's HL to stack
48A4 D5         PUSH    DE          Caller's DE to stack
48A5 C5         PUSH    BC          Caller's BC to stack
48A6 E5         PUSH    HL          Create a hole on the stack
48A7 21B348     LD      HL,48B3H    Exit addr to restore registers
48AA E3         EX      (SP),HL     Exit addr to stack
48AB F5         PUSH    AF          Rtn addr for caller of 4892
48AC AF         XOR     A           Signal no error
48AD C9         RET                 Return to caller of 4892
48AE F1         POP     AF          Clear rtn addr to 4892
48AF 3E26       LD      A,26H       Error code: file not open
48B1 B7         OR      A           Set status flag for error code
48B2 C9         RET                 Return to orginator of I/O call
*
*               ----  Restore Context After  ----
                        I/O Operation
*
48B3 C1         POP     BC          Restore caller's BC
48B4 D1         POP     DE          Restore caller's DE
48B5 E1         POP     HL          Restore caller's HL
48B6 DDE1       POP     IX          Restore caller's IX
48B8 C9         RET                 Return to I/O caller
*
*             ----  Test for end of File  ----
*
*               Status codes returned:
*                       A = 0, not end of file
*                       A = 1C, end of file
*                       A = 1D, beyond end of file
*
48B9 DD6E0A     LD      L,(IX+0AH)  L = next record no. LSB
48BC DD660B     LD      H,(IX+0BH)  H = next record no. MSB
48BF DD4E05     LD      C,(IX+05H)  C = offset to current record
48C2 7C         LD      A,H         A = MSB of next record no.
48C3 DDBE0D     CP      (IX+0DH)    Compare to MSB of ending record
48C6 200F       JR      NZ,48D7H    Jump if not near end of record
48C8 7D         LD      A,L         L = LSB of next record no.
48C9 DDBE0C     CP      (IX+0CH)    Compare to LSB of ending record
48CC 2009       JR      NZ,48D7H    Jump if not end of file
48CE 0D         DEC     C           Adjust offset to current record
48CF CD974B     CALL    4B97H       Test for EOF in sector buffer
48D2 2003       JR      NZ,48D7H    Jump if not end of file
48D4 F61C       OR      1CH         Rtn with A = 1C, (end of file flag)
48D6 C9         RET                 Return to caller
48D7 3E1D       LD      A,1DH       Beyond EOF status error code
48D9 D0         RET     NC          Return if past end of file
48DA AF         XOR     A           Signal no EOF condition
48DB C9         RET                 And return
*
*         ----  Locate GAP for Current Record Number  ----
*
48DC DD6E0A     LD      L,(IX+0AH)  L = LSB of next record number
48DF DD660B     LD      H,(IX+0BH)  H = MSB of next record number
48E2 3E05       LD      A,05H       Modulo no.
48E4 CD844B     CALL    4B84H       Divide sector modulo 5 to get
                                    relative granule
48E7 32AA49     LD      (49AAH),A   Gives relative  granule number (HL)
                                    and sector number in granule (A)
48EA DDE5       PUSH    IX          DCB addr to stack
48EC E3         EX      (SP),HL     HL = DCB addr, stack = relative
                                    granule number sought
48ED 010F00     LD      BC,000FH    Offset to 1st GAP in DCB
48F0 09         ADD     HL,BC       Get DCB (15) in HL (start of GAPs)
48F1 C1         POP     BC          BC = Q = (sector (log rec no.))/5
                                    = relative granule number for record
48F2 3E05       LD      A,05H       Number of GAP pairs to search
48F4 110000     LD      DE,0000H    Starting cumlative granule no.
*
*         ----  Top of Loop for Searching GAP's  ----
*
48F7 F5         PUSH    AF          Save count of GAP's tested
48F8 2B         DEC     HL          Backspace to 1st byte of GAP
48F9 7E         LD      A,(HL)      Load byte 1 of 1st extent pair
                                    (track no.) to test for end of GAP's
48FA 23         INC     HL          Bump to 2nd byte of extent pair,
                                    no. of consecutive granules assigned
48FB 3C         INC     A           Inc track to test for end of GAP
48FC 280B       JR      Z,4909H     Jump if end of extent (255)
48FE E5         PUSH    HL          DCB + 15 + i (i = 0, 1, 2, 3, 4) to
                                    stack (current GAP address)
48FF 62         LD      H,D         DE = Cumulative granule total
4900 6B         LD      L,E         L = LSB of granule total
4901 AF         XOR     A           Clear status (CARRY)
4902 ED42       SBC     HL,BC       Compare requested granule
                                    no. to total granules defined
                                    by GAP's thus far
4904 380F       JR      C,4915H     Jump if requested relative granule
                                    number greater than total granules
                                    counted thus far
4906 E1         POP     HL          Restore DCB addr. We have the GAP
4907 2861       JR      Z,496AH     Go compute absolute track and sector
                                    number if requested granule
                                    is in range of current GAP's
4909 23         INC     HL          Bump to 3 rd byte of extent set
490A F1         POP     AF          Get count of extent sets examined
490B 3D         DEC     A           Loop control. Count 1 set tested
490C 281A       JR      Z,4928H     Jump if all sets in this directory
                                    entry tested and GAP for this
                                    granule not found
490E 5E         LD      E,(HL)      Else, get the cumulative
                                    granule count up through
490F 23         INC     HL          this gap in DE and resume testing
4910 56         LD      D,(HL)      D = 8 MSBits of granule count
4911 23         INC     HL          Bump to first byte of next GAP
4912 23         INC     HL          Bump to second byte of next GAP
4913 18E2       JR      48F7H       And continue scanning
GAP's
*
*  ----  Come Here if the Relative Granule Number  ----
*    is Greater Than the Sum of Granules Accumulated
*
4915 24         INC     H           Gives 1 if difference greater than 256
4916 7D         LD      A,L         L = negative of requested
                                    relative granule number
4917 E1         POP     HL          Restore addr of 2nd byte of extent
4918 20EF       JR      NZ,4909H    Go if difference between requested
                                    and cumulative granule greater than
                                    256. Granule can't be in this GAP
*
*  ----  If the difference is < 256, compare it to the  ----
*  number of granules assigned to the current GAP because
*  the relative rranule may fall within the next GAP
*
491A            PUSH    DE          Save granule total to free E reg.
491B 5F         LD      E,A         Then get neg. of requested granule
491C 7E         LD      A,(HL)      and total no. of granules -1,
                                    starting with this track
491D E61F       AND     1FH         (isolate granule count), then
491F 83         ADD     A,E         Compare requested granule
                                    to no. in this GAP
4920 7B         LD      A,E         Restore neg. of requested granule no.
                                    to A in case we found needed GAP
4921 D1         POP     DE          DE = no. of granules passed so far
4922 30E5       JR      NC,4909H    Jump if requested granule not in
                                    this GAP. Go get next GAP
4924 ED44       NEG                 Convert relative granule no. to
                                    positive equivalent
4926 1842       JR      496AH       Go compute ABS track & sector number
*
*    ----  Relative Granule not in Current GAP's  ----
*   Search  Directory Entries Belonging to File for GAP
*   containing the relative granule number being sought
*
4928 CDB549     CALL    49B5H       Search dir for GAP owning relative
                                    granule. On ret, HL = 1st byte of GAP
                                    owning the relative granule
492B C0         RET     NZ          Disk error while looking at directory
492C E5         PUSH    HL          HL = track no. for requested granule
492D 60         LD      H,B         HL = requested relative granule no.
492E 69         LD      L,C         LSB of relative granule number
492F AF         XOR     A           Clear status for arithmetic
                                    operation. DE = cumulative granule
                                    through GAP in HL
4930 ED52       SBC     HL,DE       Relative granule no. sought minus
                                    total granules addressable
                                    through this GAP
4932 7D         LD      A,L         Difference gives no. of granules
                                    in current GAP. This number will
                                    be translated into numbers of
                                    tracks that will be added to the
                                    starting track number for GAP giving
                                    track number for relative granule no.
4933 329A49     LD      (499AH),A   Used to update track address
4936 E1         POP     HL          HL = Count of granules/starting
                                    track for this granule
4937 D5         PUSH    DE          Save cumulative granule count
4938 DDE5       PUSH    IX          DCB addr to stack
493A E3         EX      (SP),HL     DCB addr to HL. Granule no.
                                    for this GAP to stack
493B 110E00     LD      DE,000EH    DE = 15 = start of index for GAP's
493E 19         ADD     HL,DE       HL = DCB (15)
493F D1         POP     DE          DE = count/track no.
*
*  --- Search DCB for GAP Matching No. Returned From 49B5 ---
*
* On return, HL holds the track and granule count owning
* the relative granule. DE contains the cumulative granule
* count addressable through this track.
*
* When control returns, the DCB is searched for a matching
* track number and granule count. If one is found, control
* goes to 4993 where the absolute track and sector number
* are computed. If the DCB does not contain the track number,
* the GAP's in the DCb are moved down one entry and the track,
* count and cumulative total returned from 49B5 are copied
* to the 1st GAP entry in the DCB.
*
4940 0605       LD      B,05H       B = no. of GAP's to examine
4942 7E         LD      A,(HL)      A = 1st byte of a GAP. Search the
                                    GAP's looking for one that matches
                                    the one in DE
4943 23         INC     HL          Bump fetch addr
4944 BB         CP      E           Do the track numbers match?
4945 2006       JR      NZ,494DH    Jump if not
4947 7E         LD      A,(HL)      Yes, fetch the 2nd byte (count)
4948 AA         XOR     D           and compare with count in D reg.
4949 E6E0       AND     0E0H        Do they match
494B 2818       JR      Z,4965H     Yes, take the jump
494D 05         DEC     B           No, have we examined all GAP's
494E 2805       JR      Z,4955H     Yes, go move GAP's in DCB down
                                    and move track/count and cumulative
                                    total returned by 49B5 to 1st
                                    GAP entry in DCB
4950 23         INC     HL          No, bump fetch addr to next GAP
4951 23         INC     HL          Skip over the two-byte totals
4952 23         INC     HL          that follow each GAP in the DCB
4953 18ED       JR      4942H       Go test next GAP
*
*         ----  Move GAP's in DCB Down one Entry  ----
*
4955 D5         PUSH    DE          Save cummulative granule count
4956 EB         EX      DE,HL       DE = end of GAP's in DCB
4957 21FCFF     LD      HL,0FFFCH   HL = -4
495A 19         ADD     HL,DE       Now HL = ending addr of GAP's in DCB
495B 010C00     LD      BC,000CH    BC = no. of bytes to move (3 sets of
                                    4 bytes each gives total of 12)
495E EDB8       LDDR                DE = addr of 5th GAP entry in DCB
```

```
4960 EB          EX      DE,HL          Move all GAP entrys down one
4961 C1          POP     BC             BC = count/track no. for
                                        requested granule
4962 D1          POP     DE             DE = requested granule no. HL =
                                        DCB addr of GAP for this granule
*
*           ----  Found Matching GAP in DCB  ----
*
4963 1825        JR      498AH          Go update DCB and compute
                                        absolute track and sector no.
4965 72          LD      (HL),D         Save granule track count in DCB
4966 EB          EX      DE,HL          HL = GAP entry, DE = DCB addr
4967 F1          POP     AF             Clear stack
4968 1829        JR      4993H          Go compute absolute disk addr
*
*       Convert Granule no. & GAP Into Absolute Track/Sector
*
496A 329A49      LD      (499A),A       Save no. of granules in GAP
496D 46          LD      B,(HL)         B = 2nd byte of extent pair (no.
                                        of consecutive sectors)
496E 2B          DEC     HL             Backspace to track number
496F 4E          LD      C,(HL)         C = 1st byte of extent pair
                                        (track no.)
4970 23          INC     HL             HL = 2nd byte of extent pair
4971 F1          POP     AF             Get count of extent pairs examined
4972 2F          CPL                    And make it negative so we can
4973 C604        ADD     A,04H          Subtract 4 from it
4975 301A        JR      NC,4991H       Jump if first or second extents
4977 3C          INC     A              Gives 4/3/2/1
4978 07          RLCA                   Times 2 gives 16/12/8/4
4979 07          RLCA                   Number of bytes to transfer
497A C5          PUSH    BC             Save target GAP (trk/count)
497B D5          PUSH    DE             Save cumulative granule total
497C EB          EX      DE,HL          DE = addr of GAP owning relative gran
497D 21FCFF      LD      HL,0FFFCH      HL = - 4
4980 19          ADD     HL,DE          HL = addr of 2nd byte of
                                        previous extent pair
4981 0600        LD      B,00H          Set BC to byte count of GAPs to move
4983 4F          LD      C,A            Down for LDDR instruction
4984 EDB8        LDDR                   Move all GAPs after target GAP down
4986 EB          EX      DE,HL          HL = ending address for target GAP
4987 D1          POP     DE             DE = cumulative granule count
4988 C1          POP     BC             BC = target GAP (track/count)
4989 00          NOP                    Unnecessary instruction
498A 70          LD      (HL),B         Granule count for target GAP
498B 2B          DEC     HL             Skip back to next entry
498C 71          LD      (HL),C         Track number for target GAP
498D 2B          DEC     HL             Backspace DCB pointer
498E 72          LD      (HL),D         MSB of cumulative granule count
498F 2B          DEC     HL             Backspace DCB pointer
4990 73          LD      (HL),E         LSB of cumulative granule count
4991 60          LD      H,B            H = cum granule count for this GAP
4992 69          LD      L,C            L = track no. for current GAP
*
*           ----  Begin Disk Address Translation  ----
*
4993 7C          LD      A,H            Now isolate the starting granule
                                        number for this gap. Then add the
                                        number of granules past
4994 07          RLCA                   the start of the GAP (409A) to the
4995 07          RLCA                   granule we want. This will give
4996 07          RLCA                   us the number of tracks past the
                                        starting track to the target granule
4997 E607        AND     07H            Isolate offset to first granule
4999 C600        ADD     A,00H          Add no. of granules past first
                                        one to target granule
499B 5F          LD      E,A            Save rel. granule no. within GAP
499C E6FE        AND     0FEH           Now, any overflow should be
                                        added to the starting track
499E 0F          RRCA                   isolate and right justify
                                        overflow for track increment
499F 32AE49      LD      (49AEH),A      Save for adding to starting track no.
49A2 7B          LD      A,E            Refetch rel. granule no. within GAP
49A3 E601        AND     01H            Isolate the odd/even bit so we
                                        can compute starting track
49A5 5F          LD      E,A            offset (even-0, odd -4)
49A6 07          RLCA                   odd/even flag * 2
49A7 07          RLCA                   * 4
49A8 83          ADD     A,E            * 5
49A9 C600        ADD     A,00H          add sector no. within granule to get
                                        absolute sector no. within track
49AB 5F          LD      E,A            E = starting sector no. on track
49AC 7D          LD      A,L            A = 1st byte of extent pair (track)
49AD C600        ADD     A,00H          Add track increment for granule
                                        number within GAP
49AF 57          LD      D,A            D = track no.
49B0 DD4E06      LD      C,(IX+06H)     C = drive number for file
49B3 AF          XOR     A              Signal good status
49B4 C9          RET                    Ret to caller. DE = track/sector
*
*     ----  Find GAP for Relative Granule no. in BC  ----
*      by searching directory entries belonging to file.
*        Read overflow directory entries if necessary.
*
*  On entry BC = granule number, on exit:
*
*      L  = track
*      H  = count of granules assigned to track
*      DE = cumulative number of granules thru this GAP
*
49B5 C5          PUSH    BC             Save requested granule no.
49B6 110000      LD      DE,0000H       Initialize count of granules skipped
49B9 DD4607      LD      B,(IX+07H)     B = sector no. for directory
                                        entry and offset
49BC 78          LD      A,B            Store sector number and
49BD 32BE4A      LD      (4ABEH),A      Offset in case GAP must be assigned
49C0 DD4E06      LD      C,(IX+06H)     C = Unit no.
49C3 CDC14A      CALL    4AC1H          Read directory entry for file
49C6 011600      LD      BC,0016H       Offset to extent pairs. HL contains
                                        address of entry
49C9 09          ADD     HL,BC          Form address of GAP extents
49CA EB          EX      DE,HL          DE = address of extent pairs
49CB C1          POP     BC             BC = requsted granule no.
49CC C0          RET     NZ             Ret if error when reading directory
49CD 1A          LD      A,(DE)         Get byte one of GAP from dir sector
49CE FEFE        CP      0FEH           Test for overflow extent pair
49D0 301F        JR      NC,49F1H       Jump if FE or FF (end of extents
                                        or overflow GAP found)
49D2 13          INC     DE             No, then bump to next byte in GAP
49D3 1A          LD      A,(DE)         Fetch count of granules
49D4 E5          PUSH    HL             Save count of granules skipped
49D5 E61F        AND     1FH            Isolate granule count -1
49D7 3C          INC     A              Bump to get true count
49D8 85          ADD     A,L            Add to count of granules passed
49D9 6F          LD      L,A            If value overflows 2*16, bump MSB
49DA 3001        JR      NC,49DDH       Skip bump if no overflow
49DC 24          INC     H              Bump MSB of granules skipped
49DD E5          PUSH    HL             Save update count of granules skipped
49DE 2B          DEC     HL             Adjust granules skipped for
                                        comparison with requested granule no.
49DF AF          XOR     A              Clear carry flag
49E0 ED42        SBC     HL,BC          Compare requested granule (BC) with
                                        granule count thru this GAP (HL)
49E2 E1          POP     HL             Restore true granule count
49E3 3004        JR      NC,49E9H       Jump if requested granule exists
                                        within this GAP range
49E5 13          INC     DE             Bump to 2nd byte of next GAP
49E6 F1          POP     AF             Clear stack
49E7 18E4        JR      49CDH          Loop till GAP or FE/FF encountered
49E9 E1          POP     HL             Clear stack (requested granule no.)
49EA EB          EX      DE,HL          HL = addr of 2nd byte for GAP entry
                                        wanted. DE = accumulated gran count
49EB 7E          LD      A,(HL)         A = count of granules assigned
                                        from this track
49EC 2B          DEC     HL             Backspace to first byte of GAP
49ED 6E          LD      L,(HL)         L = track no.
49EE 67          LD      H,A            H = count of granules assigned
49EF AF          XOR     A              Signal no error
49F0 C9          RET                    Ret to caller. Target GAP located
*
*      ----  Locate Overflow Entry or Assign a New GAP  ----
*
49F1 C5          PUSH    BC             Save requested granule no.
49F2 EB          EX      DE,HL          HL = addr of dir sector entry
                                        DE = count of granules skipped
49F3 2004        JR      NZ,49F9H       Jump if FF, GAP is assigned. Could
                                        not be EOF since earlier tests would
                                        have detected it, therfore
                                        must be a write.
49F5 23          INC     HL             Otherwise an overflow indicator
                                        (FE) was found
49F6 46          LD      B,(HL)         B = offset to sector containing
                                        overflow entry.
49F7 18C3        JR      49BCH          Go locate the overflow GAP's
*
*                    ----  Assign a GAP  ----
*
49F9 CD004A      CALL    4A00H          Assign next GAP
49FC C1          POP     BC             Restore requested granule no.
49FD C0          RET     NZ             Exit if error
49FE 18B5        JR      49B5H          Restart scan. Should find GAP
                                        just assigned and exit at 49F0
*
*
*                     ----  Granule Assignment  ----
* on entry:
*          BC = requested granule number (target granule)
*          DE = Number of granules presently assigned
*          HL = Address of file entry in directory sector
*
* This point is reached when a granule number required for a WRITE
* does not exist in the GAP chain for the file. This situation
* will occur when a file is being created. Or, if a file has 50
* sectors assigned to it, and a request to write sector 60 was
* received, for example, granules for the space between sectors
* 50 and 60 would be assigned.
*
* Assignment can take two forms. First is when a granule is assigned
* to an existing GAP. The second occurs if it is not possible
* to add granules to a GAP. This case occurs when a GAP is full, 32
* granules have been assigned, or when it is impossible to assign the
* next consecutive granule to an existing GAP. In both cases, a
* new GAP must be created and added to the DCB. If the GAP area in
* the DCB is full, an overflow directory entry is created.
*
4A00 C5          PUSH    BC             Save target granule number
4A01 DD4607      LD      B,(IX+07H)     Get overflow pointer to dir entry
4A04 DD4E06      LD      C,(IX+06H)     and drive number
4A07 CDF04A      CALL    4AF0H          Go read GAT sector into 4D00
4A0A C1          POP     BC             Target granule number
4A0B C0          RET     NZ             Ret if error while reading GAT
4A0C E5          PUSH    HL             Save address of directory sector
4A0D 60          LD      H,B            H = MSB of target granule
4A0E 69          LD      L,C            L = LSB of target granule
4A0F AF          XOR     A              Clear CARRY flag
4A10 ED52        SBC     HL,DE          Target gran minus assigned grans
4A12 44          LD      B,H            BC = Difference between requested
                                        and current granule number
4A13 4D          LD      C,L            Relative granule in last GAP
4A14 03          INC     BC             and number of grans to allocate
4A15 D1          POP     DE             Load directory sector entry
4A16 13          INC     DE             Bump to 2nd byte of last GAP entry
4A17 03          INC     BC             Bump no. of granules to allocate
4A18 DD7E07      LD      A,(IX+07H)     Get sector no. for directory entry
4A1B E607        AND     07H            Isolate then compute advanced
                                        starting position in GAT
```

```
4A1D 07          RLCA                    by multiplying directory sector
4A1E 07          RLCA                    number times 4
4A1F 6F          LD      L,A             Form beginning GAT search address
4A20 264D        LD      H,4DH           Gives HL = 4D00 +(4 * sector no.)
4A22 C5          PUSH    BC              Save no. of granules to allocate
                                         and relative granule within GAP
*
*   ----  Test for Initial GAP Assignment, or GAP Full  ----
*
4A23 7B          LD      A,E             Get LSB of current GAP addr
4A24 E61E        AND     1EH             Isolate it and test for
4A26 FE16        CP      16H             First GAP word addr
4A28 2844        JR      Z,4A6EH         Jump if first GAP assignment
4A2A 1D          DEC     E               Else backspace to
4A2B 1D          DEC     E               Previous GAP (2nd byte)
4A2C 1A          LD      A,(DE)          No. of consecutive granules -1
4A2D E61F        AND     1FH             Isolate granule count
4A2F 3C          INC     A               Plus one gives true count
4A30 4F          LD      C,A             C = number of consecutive grans
4A31 FE20        CP      20H             Test if granule count field full
4A33 281F        JR      Z,4A54H         Jump if GAP full, Assign new GAP
*
*      ----  GAP not Full, try to Assign a Granule  ----
*
4A35 1A          LD      A,(DE)          Refetch 2nd byte. Count of
                                         consecutive grans already assigned
4A36 E6E0        AND     0E0H            Isolate offset to first gran in GAP
4A38 07          RLCA                    Isolate starting sector offset and
4A39 07          RLCA                    move to bit 0. Will give a value
4A3A 07          RLCA                    Of 0 or 1 in the A-reg.
4A3B 81          ADD     A,C             Add no. of consecutive grans to get
                                         total grans assigned to GAP
4A3C 47          LD      B,A             Convert granules to tracks
4A3D CB38        SRL     B               By dividing number of grans by 2
4A3F E601        AND     01H             Isolate bit that tells us which
                                         gran must be assigned next to be
                                         consecutive with current GAP
4A41 4F          LD      C,A             Save bit mask for required gran
4A42 1B          DEC     DE              Backspace to 1st byte of GAP
                                         (starting track number)
4A43 1A          LD      A,(DE)          Get starting track no. of this GAP
4A44 13          INC     DE              DE = addr of 2nd byte of GAP
4A45 80          ADD     A,B             Add no. of consecutive tracks
                                         now assigned to GAP to get next
                                         track no. to assign. This track no.
                                         is used as index into GAP
4A46 6F          LD      L,A             Form GAT addr for needed track
4A47 264D        LD      H,4DH           Gives 4D00 + (last track assigned
                                         in current GAP)
4A49 FE23        CP      23H             Test if end of GAT reached
4A4B 3007        JR      NC,4A54H        If so, must start another GAP
*
*        ----  Test if Required Granule is Available  ----
*
4A4D 79          LD      A,C             A = bit needed in GAT entry for
                                         consecutive assignment of this track
4A4E 46          LD      B,(HL)          Fetch track available word
4A4F CD5D4B      CALL    4B5DH           Test if needed gran is available
4A52 2848        JR      Z,4A9CH         Jump if available, otherwise test
                                         for end of GAP string in dir entry
*
*                 ----  Assign a New GAP  ----
*
4A54 1C          INC     E               Bump E to point to
4A55 1C          INC     E               2nd byte of next GAP
4A56 7B          LD      A,E             Then test for end of GAP's in
                                         this directory entry.
4A57 E61E        AND     1EH             Isolate LSB of addr for next GAP
4A59 FE1E        CP      1EH             Test for 30 (last pair in an entry)
4A5B 2011        JR      NZ,4A6EH        Go if not last pair of GAP for this
                                         entry, else create new GAP entry
*
*     ----  GAP Area is Full. Create an Overflow Entry  ----
*
4A5D CDB64A      CALL    4AB6H           Rewrite GAT and directory entry
4A60 C1          POP     BC              Clear stack in case of error
4A61 C0          RET     NZ              Exit if error while writing
4A62 C5          PUSH    BC              Save target granule number
4A63 CD6B4A      CALL    4A6BH           Build overflow directory entry
4A66 C1          POP     BC              Restore target granule number
4A67 C0          RET     NZ              Go if error while creating entry
4A68 C3B549      JP      49B5H           Restart GAP assignment and
                                         return to caller at 49F0
*
*        ----  Call SYS2 to Build Overflow Entry  ----
*
4A6B 3EB4        LD      A,0B4H          Call SYS2/SYS option 3
4A6D EF          RST     28H             Build overflow directory entry
                                         Ret to 4A66
*
*         ----  Granule Assignment for new GAP  ----
*
4A6E 0602        LD      B,02H           Times to scan allocation table
4A70 7D          LD      A,L             L = current track no.
4A71 FE23        CP      23H             Test if end of GAT
4A73 3007        JR      NC,4A7CH        Yes, jump if all tracks tested
4A75 7E          LD      A,(HL)          No, fetch a GAT entry
4A76 3C          INC     A               Test if any granules available
4A77 2010        JR      NZ,4A89H        Jump if some available
4A79 2C          INC     L               Else bump to next entry
4A7A 18F4        JR      4A70H           And loop till table exhausted
4A7C 2E00        LD      L,00H           Restart scan from GAT table beginning
4A7E 10F0        DJNZ    4A70H           Have we scanned table twice?
                                         No, restart scan
4A80 C1          POP     BC              Yes, write GAT table back to disk
4A81 CDB64A      CALL    4AB6H           And updated directory entry
4A84 C0          RET     NZ              Rtn if error write writing GAT
4A85 3E1B        LD      A,1BH           Disk space full error status
4A87 B7          OR      A               Set status flag for error code
4A88 C9          RET                     And return to caller
*
*         ----  Make Initial GAP Byte 2 Assignment  ----
*
4A89 3EFF        LD      A,0FFH          Set no. of consecutive grans assigned
                                         to -1, DE = address of 2nd
                                         byte of current GAP
4A8B 12          LD      (DE),A          in 2nd byte of current GAP
4A8C 0E00        LD      C,00H           Value used to build BIT 0,B instr
4A8E 46          LD      B,(HL)          Refetch GAT entry to test Bit 0
4A8F 79          LD      A,C             Prepare to build BIT X,B instruction
4A90 CD5D4B      CALL    4B5DH           Go do it. Test for 1st or 2nd gran
                                         in track being assigned
4A93 2807        JR      Z,4A9CH         Jump if first granule assigned
                                         (starting sector no. is zero)
4A95 1A          LD      A,(DE)          Else 2nd granule in track assigned
4A96 C620        ADD     A,20H           Set Bit 5 of 2nd byte of GAP then
4A98 12          LD      (DE),A          save 2nd byte of GAP
4A99 0C          INC     C               Bump to next bit in GAT entry
                                         Must be availible granule
4A9A 18F3        JR      4A8FH           Retest. This time we should take
                                         the branch at 4A90
4A9C 79          LD      A,C             A = value for bit to be turned on
                                         0 = Bit 0, 1 = Bit 1, etc.
4A9D CD3946      CALL    4639H           Turn on bit in A reg.
4AA0 B6          OR      (HL)            Combine with rest of GAT entry
4AA1 77          LD      (HL),A          And save updated GAT entry
4AA2 1D          DEC     E               Skip back to 1st entry of GAP
4AA3 1A          LD      A,(DE)          Fetch it (track no. or FF)
4AA4 3C          INC     A               Test for new track assignment or
                                         extension to existing track
                                         (call from 4A52)
4AA5 2002        JR      NZ,4AA9H        Go if extension, else its
                                         a new track assignment
4AA7 7D          LD      A,L             LSB of current GAT address is
4AA8 12          LD      (DE),A          track no. for this GAP. Save it
4AA9 1C          INC     E               Bump to 2nd byte of GAP
4AAA 1A          LD      A,(DE)          Fetch 2nd byte
4AAB 3C          INC     A               Inc no. of assigned granules
4AAC 12          LD      (DE),A          And restore. For initial assignment
                                         (gives track/00)
4AAD C1          POP     BC              Fetch count of grans to allocate
4AAE 0B          DEC     BC              Decrement and test for through
4AAF C5          PUSH    BC              Save new count then
4AB0 78          LD      A,B             Test if we're done
4AB1 B1          OR      C               Set status flags for test
4AB2 C22C4A      JP      NZ,4A2CH        Not done, go allocate another
4AB5 C1          POP     BC              Done. Clear stack, write updated
                                         GAT and directory entry
4AB6 DD4E06      LD      C,(IX+06H)      Get drive number
4AB9 CD034B      CALL    4B03H           Write updated GAT
4ABC C0          RET     NZ              Return if error during write
4ABD 0600        LD      B,00H           B = sector number & offset (IX+7)
4ABF 1815        JR      4AD6H           Write updated directory entry
                                         and return to caller
*
*    ----  READ Directory Entry for Directory Sector in B  ----
*
4AC1 C5          PUSH    BC              Save dir sector number/unit
4AC2 D5          PUSH    DE              Preserve caller's DE
4AC3 CD1E4B      CALL    4B1EH           Get track and sector no. into DE
                                         addr of file entry into HL
4AC6 2009        JR      NZ,4AD1H        Jump if sector no. out of range
4AC8 E5          PUSH    HL              Save caller's HL
4AC9 2E00        LD      L,00H           Force buffer addr to page boundry
4ACB CD354B      CALL    4B35H           Read directory sector
4ACE E1          POP     HL              Restore caller's HL
4ACF 2802        JR      Z,4AD3H         Jump if no error during READ, else
4AD1 3E11        LD      A,11H           Signal directory read error
4AD3 D1          POP     DE              Restore caller's DE
4AD4 C1          POP     BC              And sector number/unit before
4AD5 C9          RET                     Returning to caller
*
*     ----  WRITE Directory Sector Specified in B  ----
*
4AD6 C5          PUSH    BC              Save sector number/unit
4AD7 D5          PUSH    DE              Save caller's DE
4AD8 CD1E4B      CALL    4B1EH           Get directory track/sector no. in DE
4ADB 2010        JR      NZ,4AEDH        Jump if sector no. out of range
4ADD 2E00        LD      L,00H           Force buffer addr to page boundry
4ADF CDEF46      CALL    46EFH           Write directory sector
4AE2 2007        JR      NZ,4AEBH        Jump if error during WRITE
4AE4 CDF346      CALL    46F3H           Read sector to verify write
4AE7 FE06        CP      06H             Test for good read
4AE9 2802        JR      Z,4AEDH         Jump if no error during read
4AEB 3E12        LD      A,12H           Directory write error code
4AED D1          POP     DE              Restore caller's DE
4AEE C1          POP     BC              And sector number/unit
4AEF C9          RET                     Then return to caller
*
*          ----  READ GAT Sector for Unit in C  ----
*
4AF0 D5          PUSH    DE              Save caller's DE
4AF1 E5          PUSH    HL              Save caller's HL
4AF2 CD554B      CALL    4B55H           Get directory track number into D
4AF5 1E00        LD      E,00H           Set sector no. to 0 (GAT sector)
4AF7 21004D      LD      HL,4D00H        Set buffer address = 4D00
4AFA CD354B      CALL    4B35H           Read GAT sector for unit in C
4AFD E1          POP     HL              Restore caller's HL
4AFE D1          POP     DE              Restore caller's DE
4AFF C8          RET     Z               Return if no error during read
4B00 3E14        LD      A,14H           Else signal GAT read error
4B02 C9          RET                     Then return to caller
*
*          ----  WRITE GAT Sector for Unit in C  ----
*
4B03 D5          PUSH    DE              Save caller's DE
4B04 E5          PUSH    HL              Save caller's HL
4B05 CD554B      CALL    4B55H           Get directory track number into D
4B08 1E00        LD      E,00H           E = GAT sector number
4B0A 21004D      LD      HL,4D00H        Buffer address for sector
```

```
4B0D CDEF46     CALL    46EFH       Write updated GAT sector
4B10 2007       JR      NZ,4B19H    Jump if error during write
4B12 CDF346     CALL    46F3H       Else read sector just written
4B15 FE06       CP      06H         Test for error during read
4B17 2802       JR      Z,4B1BH     Jump if no error during read
4B19 3E15       LD      A,15H       Else signal GAT write error
4B1B E1         POP     HL          Restore caller's HL
4B1C D1         POP     DE          Restore caller's DE
4B1D C9         RET                 Ret to caller w/status in A reg.
*
*           ----  Prepare Registers for Directory I/O  ----
*
*   Entry conditions : B = system sector number
*                    : C = unit number
*
*   Exit conditions  : DE = track/sector number
*                    : HL = sector buffer addr for file entry
*                    : AF = non-zero status - sector out of range
*
4B1E CD554B     CALL    4B55H       Get track number for unit in C
4B21 78         LD      A,B         B = offset for file entry in dir
4B22 E6E0       AND     0E0H        Isolate file entry number
4B24 6F         LD      L,A         LSB of buffer addr for file
4B25 2642       LD      H,42H       HL = 42XX (XX = rel offset to file)
4B27 A8         XOR     B           Isolate sector number (bits 0-4)
4B28 FE08       CP      08H         Test for sector out of range
4B2A 3005       JR      NC,4B31H    Jump if sector out of range
4B2C C602       ADD     A,02H       Form true sector number
4B2E 5F         LD      E,A         And move to E-reg
4B2F AF         XOR     A           Signal no error
4B30 C9         RET                 Ret W/HL = sector file entry addr
4B31 3E10       LD      A,10H       Error code, sector no. out of range
4B33 B7         OR      A           Set status flags for error
4B34 C9         RET                 And return to caller
*
*           ----  Subroutine for Calling Disk Driver  ----
*     Entry conditions :
*
*                     DE = track/sector number
*                      C = unit number
*                     HL = sector buffer address
*
4B35 CDDD46     CALL    46DDH       Call disk driver for read
4B38 D606       SUB     06H         Test if operation terminated normally
4B3A C8         RET     Z           Return if yes
4B3B C606       ADD     A,06H       Unncessary instruction
4B3D C0         RET     NZ          Unnecessary instruction
4B3E D5         PUSH    DE          Save disk addr where error occurred
4B3F 110000     LD      DE,0000H    Track 0 sector 0 (Boot loader addr
4B42 CDDD46     CALL    46DDH       Read track 0 Sector 0
4B45 D1         POP     DE          Restore addr of disk error
4B46 C0         RET     NZ          Return to caller with bad status
                                    if boot sector read error
4B47 E5         PUSH    HL          Save original buffer address
4B48 23         INC     HL          Bump to 2 byte
4B49 23         INC     HL          Of boot loader
4B4A 56         LD      D,(HL)      Fetch directory track number
                                    from boot loader
4B4B 3E04       LD      A,04H       Offset to directory table
4B4D 81         ADD     A,C         Add drive number to table offset
4B4E 6F         LD      L,A         Then form directory table address for
4B4F 2643       LD      H,43H       drive specified in C
4B51 72         LD      (HL),D      and save directory track number
4B52 E1         POP     HL          Restore caller's buffer address
4B53 18E0       JR      4B35H       Go retry operation
*
*     ----   Return Directory Track Number for Unit C  ----
*
4B55 3E04       LD      A,04H       Offset to directory table
4B57 81         ADD     A,C         Add drive number to offset
4B58 6F         LD      L,A         Then form directory table address
4B59 2643       LD      H,43H       For drive in HL
4B5B 56         LD      D,(HL)      Get dir track no. for unit in C
4B5C C9         RET                 Return to caller
*
*
* Build and execute an instruction to test the bit given by the value
* in the A-register for the quantity in the B-register.
*
4B5D E607       AND     07H         Isolate bit position to be tested
4B5F 07         RLCA                Left shift
4B60 07         RLCA                Bit value three
4B61 07         RLCA                places before
4B62 F640       OR      40H         Build 2nd half of BIT (A),B instr
4B64 32684B     LD      (4B68H),A   Save 2nd half of instruction
4B67 CB40       BIT     00H,B       Then execute it
4B69 C9         RET                 and return to caller
*
* Compute sector number for record number in DE. Record length
* in A. Sector number = (rec.size (A) * record number (DE)/256.
* Called from POSN
*
4B6A C5         PUSH    BC          Save caller's BC
4B6B EB         EX      DE,HL       Record number to DE
4B6C 4F         LD      C,A         Record length to C
4B6D 210000     LD      HL,0000H    Set up to multiply record length
                                    by record size using shift add method
4B70 7D         LD      A,L         Initialize product as zero
4B71 0608       LD      B,08H       Maximum shift count
4B73 29         ADD     HL,HL       Shift prod. left for each bit tested
4B74 17         RLA                 Shift bits into CARRY for ADC below
4B75 CB01       RLC     C           They will migrate into the product
                                    Test for 'ON' bit in record length
4B77 3003       JR      NC,4B7CH    Jump if no 'ON' bit
4B79 19         ADD     HL,DE       Add record number to product
4B7A CE00       ADC     A,00H       Pick up any possible CARRY from L
4B7C 10F5       DJNZ    4B73H       Loop till 8 record length bits tested
4B7E 4F         LD      C,A         MSB of product
4B7F 7D         LD      A,L         Remainder (offset in sector of record
4B80 6C         LD      L,H         LSB of product
4B81 61         LD      H,C         MSB of product
4B82 C1         POP     BC          Restore caller's BC
4B83 C9         RET                 Return to caller
*
* Modulo division routine on entry: HL = dividend, A = divisor.
* Upon exit HL = quotient, A = remainder. Shift the most significant
* bit out of HL into the A-reg one bit at a time. Whenever the A-reg
* is >= the modulus, or full,  decrement it by the modulo * value
* and bump the quotient.
*
4B84 D5         PUSH    DE          Save caller's DE
4B85 57         LD      D,A         D = divisor
4B86 1E10       LD      E,10H       Maximum number of bits to test
4B88 AF         XOR     A           Zero A, CARRY
4B89 29         ADD     HL,HL       Shift dividend left 1 bit
                                    overflow to CARRY
4B8A 17         RLA                 CARRY to Bit 0. Bit 7 to CARRY
4B8B 3803       JR      C,4B90H     Jump if remainder is large
4B8D BA         CP      D           Else compare it to the modulus
4B8E 3802       JR      C,4B92H     Jump if remainder <= to modulus
4B90 92         SUB     D           Else, adjust remainder
4B91 2C         INC     L           And quotient
4B92 1D         DEC     E           All bits been tested?
4B93 20F4       JR      NZ,4B89H    Jump if not
4B95 D1         POP     DE          Yes, restore caller's DE
4B96 C9         RET                 and return
*
*
*
4B97 DD7E08     LD      A,(IX+08H)  Fetch offset EOF from DCB
4B9A 3D         DEC     A           Adjust offset by one
4B9B 91         SUB     C           Compare to current record offset
4B9C 3F         CCF                 Invert CARRY for test beyond EOF
4B9D 03         INC     BC          Readjust offset
4B9E C9         RET                 and return to caller
4B9F 00         NOP                 Inc P-register for assembler
*
*
*
4BA0 AF         XOR     A           Signal good status
4BA1 C9         RET                 And return to caller
*
*         ----  RST 28 Processing. Load System Overlay ----
*
4BA2 E3         EX      (SP),HL     Save ret addr for RST 28 in HL
4BA3 E1         POP     HL          Restore HL, return addr lost
4BA4 B7         OR      A           Test overlay call request
4BA5 F21243     JP      P,4312H     Invalid request if positive
4BA8 E5         PUSH    HL          Save caller's HL
4BA9 67         LD      H,A         Save overlay request
4BAA 3A0E43     LD      A,(430EH)   Fetch last overlay request code
4BAD AC         XOR     H           And compare
4BAE E60F       AND     0FH         With current
4BB0 7C         LD      A,H         Restore current request
4BB1 320E43     LD      (430EH),A   And save as previous also
4BB4 2837       JR      Z,4BEDH     Jump if overlay already in memory
4BB6 D5         PUSH    DE          Save caller's DE
4BB7 C5         PUSH    BC          and BC
4BB8 E60F       AND     0FH         Isolate directory sector number
4BBA FE08       CP      08H         Test for 1st or 2nd entry in sector
4BBC 3802       JR      C,4BC0H     Jump if 2st entry to be used
4BBE C618       ADD     A,18H       Else get new sec no. - file offset
4BC0 47         LD      B,A         Move to B. Will be used for
                                    computing buffer addr by 4B1E
4BC1 32A744     LD      (44A7H),A   Save in DCB as record no. too
4BC4 AF         XOR     A           Clear CARRY and A-reg
4BC5 32A144     LD      (44A1H),A   Grant all permissions
4BC8 ED62       SBC     HL,HL       Generate 16 Bits of zero
4BCA 22AA44     LD      (44AAH),HL  And save as next record number
4BCD 4F         LD      C,A         Set drive number to zero
4BCE CDC14A     CALL    4AC1H       Read directory sector into 4200
4BD1 C20944     JP      NZ,4409H    Jump if error during read
4BD4 7D         LD      A,L         LSB of file entry addr in buffer
4BD5 C616       ADD     A,16H       Add offset to position to first GAP
4BD7 6F         LD      L,A         Form GAP address in HL
4BD8 7E         LD      A,(HL)      Fetch track number for overlay
4BD9 2C         INC     L           Bump to 2nd GAP byte
4BDA 66         LD      H,(HL)      Fetch sector offset/granules assigned
4BDB 6F         LD      L,A         HL=1st GAP for overlay
4BDC 22AE44     LD      (44AEH),HL  Save as 1st GAP in short system DCB
4BDF 11A044     LD      DE,44A0H    Address of short system DCB
4BE2 CD394C     CALL    4C39H       Go load file from system DCB
4BE5 C20944     JP      NZ,4409H    Jump if any error during load
4BE8 22FC4B     LD      (4BFCH),HL  Save transfer address of overlay
4BEB C1         POP     BC          Restore caller's BC
4BEC D1         POP     DE          DE
4BED E1         POP     HL          And HL
4BEE 3A1543     LD      A,(4315H)   Fetch DEBUG flag and save it
4BF1 32004C     LD      (4C00H),A   for recall after overlay execution
4BF4 AF         XOR     A           Zero A-reg
4BF5 321543     LD      (4315H),A   Clear DEBUG flag
4BF8 3A0E43     LD      A,(430EH)   Load overlay request code
4BFB CD0000     CALL    0000H       Enter overlay
4BFE F5         PUSH    AF          Save so we can use A-reg
4BFF 3E00       LD      A,00H       Restore DEBUG
4C01 321543     LD      (4315H),A   Flag to 4315
4C04 F1         POP     AF          Restore overlay exit status
4C05 C9         RET                 Return to overlay caller
*
*           ----  Load File From System DCB  ----
*
4C06 E5         PUSH    HL          Save caller's HL
4C07 CD3044     CALL    4430H       Load file specified by system DCB
4C0A C20944     JP      NZ,4409H    Jump if error during load
4C0D E3         EX      (SP),HL     Restore HL. Save stack contents
4C0E 3AOF43     LD      A,(430FH)   Fetch DEBUG flag
4C11 B7         OR      A           Set status flags for DEBUG
4C12 FA0F40     JP      M,400FH     If DEBUG active, go load it
4C15 C9         RET                 Return to caller
```

```
*
*          ----  Load File From DCB Specified in DE  ----
*
4C16 0600       LD      B,00H           Specify physical I/0
4C18 21004D     LD      HL,4D00H        Buffer address
4C1B CD2444     CALL    4424H           OPEN file
4C1E 2809       JR      Z,4C29H         Jump if error during OPEN
4C20 CBF7       SET     06H,A           Convert possible error code of 18-58
4C22 FE58       CP      58H             Was error code 18
4C24 C0         RET     NZ              Return if no
4C25 3E5F       LD      A,5FH           Yes, replace it with a 1F
4C27 B7         OR      A               Set status flags
4C28 C9         RET                     And return to caller
*
*             ----  General Purpose File Loader  ----
*
4C29 13         INC     DE              Bump DCB pointer to access flags
4C2A 1A         LD      A,(DE)          Fetch file status flags
4C2B 1B         DEC     DE              Backspace DCB pointer
4C2C E607       AND     07H             Isolate access permission
4C2E FE06       CP      06H             Test for restricted file
4C30 3807       JR      C,4C39H         Jump if file not restricted
4C32 3A0F43     LD      A,(430FH)       Fetch DEBUG flag
4C35 07         RLCA                    DEBUG active bit to CARRY
4C36 3E25       LD      A,25H           Error code if DEBUG active
4C38 D8         RET     C               Return with error if DEBUG active
*
*                   ----  System Loader  ----
*
4C39 01FF4D     LD      BC,4DFFH        Set BC to empty buffer status
4C3C CD8C4C     CALL    4C8CH           Get a byte from load file
4C3F FE01       CP      01H             Test for control code
4C41 2824       JR      Z,4C67H         Jump if load data follows
4C43 FE02       CP      02H             Test for control code 2
4C45 2813       JR      Z,4C5AH         Jump if transfer address follows
4C47 FE20       CP      20H             Test for illegal control code
4C49 3804       JR      C,4C4FH         Jump if illegal control code,
                                        skip to next field
4C4B 3E22       LD      A,22H           Else ret LOAD FILE FORMAT error
4C4D B7         OR      A               Set status for error code
4C4E C9         RET                     Return to file load caller
*
*             ----  Skip to Next Control Byte  ----
*
4C4F CD8C4C     CALL    4C8CH           Get byte count for this control code
4C52 47         LD      B,A             for DJNZ instruction
4C53 CD8C4C     CALL    4C8CH           Fetch a byte
4C56 10FB       DJNZ    4C53H           Loop till all bytes for current
                                        control code skipped
4C58 18E2       JR      4C3CH           Go fetch next control code
*
*               ----  Get Transfer Address Into HL  ----
*
4C5A CD8C4C     CALL    4C8CH           Skip over byte count
4C5D CD8C4C     CALL    4C8CH           Get LSB of transfer address
4C60 6F         LD      L,A             Move to L
4C61 CD8C4C     CALL    4C8CH           Get MSB of transfer address
4C64 67         LD      H,A             Form address in HL
4C65 AF         XOR     A               Signal good load
4C66 C9         RET                     Return to file load caller
*
*             ----  Load Data Into Memory  ----
*
4C67 CD8C4C     CALL    4C8CH           Get count of load bytes
4C6A 47         LD      B,A             for DJNZ instruction
4C6B CD8C4C     CALL    4C8CH           Get LSB of load address
4C6E 6F         LD      L,A             Move it to L
4C6F 05         DEC     B               Count 1 byte
4C70 CD8C4C     CALL    4C8CH           Get MSB of load address
4C73 67         LD      H,A             Form load address in HL
4C74 05         DEC     B               Count 1 byte
4C75 CD8C4C     CALL    4C8CH           Fetch a byte of load data
4C78 77         LD      (HL),A          Save it in memory
4C79 BE         CP      (HL)            Make sure it was stored
4C7A 2005       JR      NZ,4C81H        Go if not, attempt to store in ROM
4C7C 23         INC     HL              Good store. Bump store address
4C7D 10F6       DJNZ    4C75H           Loop till data block stored
4C7F 18BB       JR      4C3CH           Go get next control byte
*
*             ----  Determine Cause of Storage Failure  ----
*
4C81 7E         LD      A,(HL)          Fetch original contents of memory
4C82 2F         CPL                     Complement it and
4C83 77         LD      (HL),A          Store it back. Then
4C84 BE         CP      (HL)            Test if byte was stored
4C85 3E63       LD      A,63H           Error code for MEMORY FAULT
4C87 C0         RET     NZ              Exit if memory error
4C88 3E64       LD      A,64H           Else ret code for attempted to load ROM
4C8A B7         OR      A               Set status flags for error code
4C8B C9         RET                     Return to file load caller
*
*    ----  Get Next Byte of Load Data From Sector Buffer  ----
*
4C8C 0C         INC     C               Bump to next byte in sector buffer
4C8D 2014       JR      NZ,4CA3H        Jump if buffer not empty
4C8F D5         PUSH    DE              DCB address to stack
4C90 DDE3       EX      (SP),IX         Then move it to IX for read routine
4C92 E5         PUSH    HL              Save caller's HL
4C93 D5         PUSH    DE              And DE
4C94 C5         PUSH    BC              And buffer address
4C95 CD1B48     CALL    481BH           Go read next sector from file
4C98 C1         POP     BC              Restore buffer address
4C99 D1         POP     DE              And DE
4C9A E1         POP     HL              And HL
4C9B DDE1       POP     IX              Clear stack
4C9D 2804       JR      Z,4CA3H         Jump if no error during read
4C9F C1         POP     BC              Clear 4C8C (ret address) from stack
4CA0 F640       OR      40H             Signal error during load
4CA2 C9         RET                     Return to file load caller
*
*  Re-initialize BC to 4D00 and fetch first byte of data from buffer
*
4CA3 C5         PUSH    BC              Remove buffer address of 4D00
                                        (unnecessary)
4CA4 064D       LD      B,4DH           Force address to 4D00
                                        (unnecessary)
4CA6 0A         LD      A,(BC)          Fetch 1st data byte
4CA7 C1         POP     BC              Restore 4D00
4CA8 C9         RET                     Return to caller
*
*             ----  CLOCK Display Routine  ----
*
4CA9 AC4C       DEFW    4CAC5
4CAB 05         DEFB    5               200 millisecond counter
4CAC DD3502     DEC     (IX+02H)        Count a 200 MS interval
4CAF C0         RET     NZ              Exit if 1 second not elapsed
4CB0 DD360205   LD      (IX+02H),05H    Reset millisec counter to 1 second
4CB4 21353C     LD      HL,3C35H        Video display address for time
4CB7 114340     LD      DE,4043H        End address of max time values
4CBA 0E3A       LD      C,3AH           Hex value for : (field separator)
4CBC 0603       LD      B,03H           No. of values to convert to ASCII
4CBE 1A         LD      A,(DE)          Get a binary time value starting
                                        with hour, minute, second
4CBF 1B         DEC     DE              Backspace to next time value
4CC0 362F       LD      (HL),2FH        Convert binary to ASCII by compound
4CC2 34         INC     (HL)            Subtraction. Begin with quotient of 2F
4CC3 D60A       SUB     0AH             and increment to 30, 31, ... Divide
4CC5 30FB       JR      NC,4CC2H        value by 10 using subtraction
                                        Loop till division complete
4CC7 C63A       ADD     A,3AH           Force remainder positive
4CC9 23         INC     HL              Bump to next location in display
4CCA 77         LD      (HL),A          Save 2nd digit (remainder)
4CCB 23         INC     HL              Bump to next position in display
4CCC 05         DEC     B               Have 3 digits been displayed
4CCD C8         RET     Z               Return if yes
4CCE 71         LD      (HL),C          No, display : following value
4CCF 23         INC     HL              Bump to next location in display
4CD0 18EC       JR      4CBEH           Loop till 3 values displayed
4CD2 114640     LD      DE,4046H        Ending address of binary date
4CD5 0E2F       LD      C,2FH           ASCII value for / (field separator)
4CD7 18E3       JR      4CBCH           Use time conversion routine
*
*             ----  TRACE Display Routine  ----
*
4CD9 DB4C       DEFW    4CDBH
4CDB 211200     LD      HL,0012H        Offset to P-reg in stack frame
4CDE 39         ADD     HL,SP           Backspace stack pointer
4CDF 5E         LD      E,(HL)          LSB of P-reg at interrupt time
4CE0 23         INC     HL              Backspace one more byte
4CE1 56         LD      D,(HL)          MSB of P-register at interrupt time
4CE2 212E3C     LD      HL,3C2EH        Display address for TRACE display
4CE5 7A         LD      A,D             Fetch MSB of P-reg
4CE6 CDEA4C     CALL    4CEAH           Convert to ASCII and display
4CE9 7B         LD      A,E             LSB to ASCII and display
4CEA F5         PUSH    AF              Save value to be converted
4CEB 1F         RRA                     Right shift it
4CEC 1F         RRA                     4 bits to isolate
4CED 1F         RRA                     upper digit of lower
4CEE 1F         RRA                     nibble, convert to ASCII and display
4CEF CDF34C     CALL    4CF3H           Convert upper digit of nibble
                                        to ASCII and display
4CF2 F1         POP     AF              Restore original nibble
4CF3 E60F       AND     0FH             Isolate lower digit
4CF5 C630       ADD     A,30H           Convert binary to ASCII
4CF7 FE3A       CP      3AH             Test for hex values A - F
4CF9 3802       JR      C,4CFDH         Jump if not A - F
4CFB C607       ADD     A,07H           Else convert to proper ASCII value
4CFD 77         LD      (HL),A          Send ASCII digit to video
4CFE 23         INC     HL              Bump to next video address
4CFF C9         RET                     Return to caller
*
*               ----  System Sector Buffer  ----
*
4D00            DEFS    256             System sector buffer area
*
*             ----  System Initialization Code  ----
*
4E00 F3         DI                      Kill interrupts
4E01 ED56       IM      1               Set interrupt mode 1 (RST 38)
4E03 31FC41     LD      SP,41FCH        System stack area
4E06 210052     LD      HL,5200H        Utility load
4E09 224740     LD      (4047H),HL      Address to 4047
4E0C 21FFFF     LD      HL,0FFFFH       Start addr for memory size test
4E0F 7E         LD      A,(HL)          Save memory contents
4E10 47         LD      B,A             Save for restoration
4E11 2F         CPL                     Modify memory value, then
4E12 77         LD      (HL),A          Put it back and compare stored
4E13 BE         CP      (HL)            location with register value
4E14 70         LD      (HL),B          Restore orginal memory contents
4E15 2806       JR      Z,4E1DH         Jump if memory location exists
4E17 7C         LD      A,H             Else get failed addr and lower
4E18 D604       SUB     04H             by 1024 and try again
4E1A 67         LD      H,A             Form new store address
4E1B 18F2       JR      4E0FH           aad repeat test
4E1D 224940     LD      (4049H),HL      Save max memory address
4E20 C3204F     JP      4F20H           Go change keyboard driver addr
4E23 22B843     LD      (43B8H),HL      and save new driver addr
4E26 22C243     LD      (43C2H),HL      in Driver Address Table
4E29 2A1B40     LD      HL,(401BH)      Addr of keyboard buffer
4E2C 22C043     LD      (43C0H),HL      to Device Table
4E2F 2A1E40     LD      HL,(401EH)      Addr of video driver
4E32 22BA43     LD      (43BAH),HL      to Driver Table
4E35 22C643     LD      (43C6H),HL      and Driver Address Table
4E38 2A2340     LD      HL,(4023H)      Video buffer address
4E3B 22C443     LD      (43C4H),HL      to Device Table
4E3E 2A2640     LD      HL,(4026H)      Printer driver address
4E41 22BC43     LD      (43BCH),HL      to Driver Address Table
4E44 22CA43     LD      (43CAH),HL      and Device Table
4E47 2A2B40     LD      HL,(402BH)      Printer buffer address
```

```
4E4A 22C843     LD      (43C8H),HL    to Device Table
4E4D 21CC43     LD      HL,43CCH      End address of Device Table
4E50 3600       LD      (HL),00H      Terminate list
4E52 2C         INC     L             Unnecessary instruction
4E53 2000       JR      NZ,4E55H      Unnecessary instruction
4E55 3E03       LD      A,03H         Unnecessary instruction
4E57 211540     LD      HL,4015H      Address of keyboard DCB
4E5A 115843     LD      DE,4358H      Addr of alternate DCB area
4E5D F5         PUSH    AF            Unnecessary instruction
4E5E 010800     LD      BC,0008H      Number of bytes to move
4E61 EDB0       LDIR                  Move keyboard DCB to alternate area
4E63 AF         XOR     A             Zero A-register
4E64 0608       LD      B,08H         Number of bytes to zero
4E66 12         LD      (DE),A        Zero 8 bytes after alternate DCB
4E67 13         INC     DE            Bump to next address
4E68 10FC       DJNZ    4E66H         Loop till 8 bytes zeroed
4E6A 3EFF       LD      A,0FFH        Data storage pattern
4E6C 0610       LD      B,10H         Number of bytes to store
4E6E 12         LD      (DE),A        Store FF from 4368 to 4377
4E6F 13         INC     DE            Bump store address
4E70 10FC       DJNZ    4E6EH         Loop till 32 bytes stored
4E72 F1         POP     AF            Unnecessary instruction
4E73 3D         DEC     A             Unnecessary instruction
4E74 2000       JR      NZ,4E76H      Unnecessary instruction
4E76 3AEC37     LD      A,(37ECH)     Unnecessary instruction
4E79 AF         XOR     A             Load op code for a NOP
4E7A 321543     LD      (4315H),A     Store NOP at DEBUG switch
4E7D 214C40     LD      HL,404CH      Address of interrupt mask byte
4E80 77         LD      (HL),A        Zero it (Ignore all interrupts)
4E81 3EC3       LD      A,0C3H        JP op code
4E83 111845     LD      DE,4518H      Addr of interrupt routine
4E86 ED531340   LD      (4013H),DE    Form JP  4518 instruction at 4012
4E8A 321240     LD      (4012H),A     Save JP  op code
4E8D 116045     LD      DE,4560H      Store addr of clock interrupt
4E90 ED535B40   LD      (405BH),DE    Routine in Interrupt Task List
4E94 CBFE       SET     07H,(HL)      Set mask bit for clock interrupt
4E96 116946     LD      DE,4669H      Address of disk interrupt routine
4E99 ED535940   LD      (4059H),DE    To Interrupt Task List
4E9D CBF6       SET     06H,(HL)      Set mask bit for disk interrupt
4E9F 3E01       LD      A,01H         Unnecessary instruction
4EA1 320F43     LD      (430FH),A     Unnecessary instruction
4EA4 11AF45     LD      DE,45AFH      Address of clock maint. routine
4EA7 3E07       LD      A,07H         Index into Interrupt Task List
4EA9 CD1044     CALL    4410H         Add clock maint routine to task list
4EAC 21ED4E     LD      HL,4EEDH      Address of TRSDOS SYSTEM message
4EAF 7E         LD      A,(HL)        Fetch a character from message
4EB0 ED44       NEG                   Decode it
4EB2 CD3300     CALL    0033H         Display it
4EB5 23         INC     HL            Bump to next byte of message
4EB6 FE0D       CP      0DH           Test for end of message
4EB8 20F5       JR      NZ,4EAFH      Loop till message displayed
4EBA CD2B00     CALL    002BH         Read the keyboard
4EBD FE0D       CP      0DH           Test if carriage return active
4EBF CA0044     JP      Z,4400H       Jmp if yes, skip AUTO test
4EC2 0E00       LD      C,00H         Prepare to read GAT sector and
4EC4 1611       LD      D,11H         Test for AUTO procedure
4EC6 1E00       LD      E,00H         DE = track/sector number
4EC8 210042     LD      HL,4200H      HL = buffer address
4ECB CD354B     CALL    4B35H         Read GAT sector
4ECE C20944     JP      NZ,4409H      Jump if any error during read
4ED1 3AE042     LD      A,(42E0H)     Fetch 1st byte of AUTO procedure
4ED4 FE0D       CP      0DH           File and test for file name
4ED6 CA0044     JP      Z,4400H       Jump if no file specified
4ED9 21E042     LD      HL,42E0H      Address of AUTO file specification
4EDC 111843     LD      DE,4318H      Addr of command line buffer
4EDF 012000     LD      BC,0020H      Number of bytes in AUTO file name
4EE2 EDB0       LDIR                  Move name to command line buffer
4EE4 211843     LD      HL,4318H      Restore addr of command line buffer
4EE7 CD6744     CALL    4467H         Display AUTO command
4EEA C30544     JP      4405H         Load SYS1 to execute command
*
*               ---- System Header Message ----
*
4EED            DEFM    'TRSDOS - DISK OPERATING SYSTEM - VER 2.3'
4F17 F3         DEFB    0F3H          Message terminator (carriage return)
*
*
*
4F20 21D843     LD      HL,43D8H      Addr of replacement keyboard driver
4F23 221640     LD      (4016H),HL    To keyboard DCB
4F26 C3234E     JP      4E23H         Rejoin main code
```

```
*************************************************************
*                                                           *
*                        SYS1/SYS                           *
*                 Command Line Interpreter                  *
*                                                           *
*************************************************************


4E00 E670        AND     70H             Isolate option
4E02 FE10        CP      10H             Test for option 10
4E04 CA1F4E      JP      Z,4E1FH         Read next command and process
4E07 FE20        CP      20H             Test for option 20
4E09 2814        JR      Z,4E1FH         Read next command and process
4E0B FE30        CP      30H             Test for option 30
4E0D 283F        JR      Z,4E4EH         Reprocess command line buffer
4E0F FE40        CP      40H             Test for option 40
4E11 CA564F      JP      Z,4F56H         Copy and Edit file name
4E14 FE50        CP      50H             Test for option 50
4E16 CAC04F      JP      Z,4FC0H         Add suffix to file name
4E19 FE60        CP      60H             Test for option 60
4E1B CA7D50      JP      Z,507DH         Parse command list
4E1E C9          RET                     Unknown option. Return to caller
*
*      --- Begin processing for options 10 and 20 ---
*
4E1F FB          EI                      Allow interrupts
4E20 31FC41      LD      SP,41FCH        Reset stack pointer
4E23 215943      LD      HL,4359H        This  instruction not used
4E26 CB6E        BIT     05H,(HL)        This instruction not used
4E28 2A0F43      LD      HL,(430FH)      System condition flag
4E2B CB7E        BIT     07H,(HL)        Is DEBUG active
4E2D 2805        JR      Z,4E34H         Jump if no
4E2F 3EC3        LD      A,0C3H          Else, construct a
4E31 321543      LD      (4315H),A       JP XXXX instruction at 4315
4E34 CB6E        BIT     05H,(HL)        Test for program chaining
4E36 2808        JR      Z,4E40H         Jump if no
4E38 3A4038      LD      A,(3840H)       Get keyboard row - 0
4E3B E604        AND     04H             Test for break
4E3D C23040      JP      NZ,4030H        Go recall SYS1 with option 2 if so
4E40 21B851      LD      HL,51B8H        Address of DOS READY message
4E43 CD6744      CALL    4467H           Write DOS READY to video
4E46 211843      LD      HL,4318H        Address of command line buffer
4E49 063F        LD      B,3FH           Max no. of key strokes allowed
4E4B CD4000      CALL    0040H           Get next command from keyboard
*
*          --- Begin processing for option 40 ---
*
4E4E 012D40      LD      BC,402DH        Return point in SYS0
4E51 C5          PUSH    BC              To stack
4E52 118044      LD      DE,4480H        Address of system DCB
4E55 CD564F      CALL    4F56H           Move file name to DCB (command line)
4E58 C2874E      JP      NZ,4E87H        Go if illegal char. found
4E5B 1A          LD      A,(DE)          Fetch first character of command
4E5C FE2A        CP      2AH             Test for *
4E5E CA874E      JP      Z,4E87H         Go if command starts with * (error)
4E61 E5          PUSH    HL              Save end position in command list
4E62 01A44E      LD      BC,4EA4H        Address of SYS1 commands
4E65 CD2A50      CALL    502AH           Test for SYS1
cmd BASIC2/DEBUG/TRACE
4E68 281A        JR      Z,4E84H         Jump if SYS1 command. Execute CMD
4E6A 01BD4E      LD      BC,4EBDH        Else load addr. of other DOS commands
4E6D CD2A50      CALL    502AH           Compare current command against
                                         list of SYS6 commands
4E70 2812        JR      Z,4E84H         Jump if a match found
4E72 3A0F43      LD      A,(430FH)       Fetch system condition flag
4E75 CBA7        RES     04H,A           Clear SYS6 loaded flag
*
*         Command is neither SYS 1 nor SYS 6 command.
*      Add CMD suffix to command and load file from disk.
*
4E77 320F43      LD      (430FH),A       Restore updated condition flag
4E7A 21B551      LD      HL,51B5H        Address of CMD message
4E7D CDC04F      CALL    4FC0H           Add CMD suffix to file name
4E80 E1          POP     HL              Clear DCB address from stack
4E81 C33344      JP      4433H           Enter routine to load and execute file
4E84 E1          POP     HL              Clear stack
4E85 D5          PUSH    DE              SYS1 addr for DOS cmd to stack
4E86 C9          RET                     Go to SYS1 addr for DOS command.
                                         (514C, 5162, 5191, or 4E90)
*
*
*
4E87 21C351      LD      HL,51C3H        Address of "WHAT" message
4E8A CD6744      CALL    4467H           Print "WHAT"
4E8D C33040      JP      4030H           Go wait for next command
4E90 3A0F43      LD      A,(430FH)       Get system condition flags
4E93 CB67        BIT     04H,A           Test if SYS6 currently loaded
4E95 C20052      JP      NZ,5200H        Jump if yes
4E98 3A0F43      LD      A,(430FH)       Else reload system condition flag
4E9B CBE7        SET     04H,A           And clear SYS6 loaded flag
4E9D 320F43      LD      (430FH),A       Restore system condition flags
4EA0 3E88        LD      A,88H           DOS overlay req. for SYS6 option 0
4EA2 F5          PUSH    AF              Dummy push for overlay loader
4EA3 EF          RST     28H             Load and execute SYS6
*
*            Text Strings for SYS1 Commands
*
4EA4             DEFM    'BASIC2'        Text for SYS1 cmd
4EAA             DEFW    514CH           Addr of BASIC2 cmd in SYS1
4EAC             DEFM    'DEBUG'         Text for SYS1 cmd
4EB2             DEFW    5162H           Addr of DEBUG cmd in SYS1
4EB4             DEFM    'TRACE'         Text for SYS1 cmd
4EBA             DEFW    5191H           Addr of TRACE cmd in SYS1
4EBC             DEFB    00H             Text sting terminator
*
*            Text Strings for SYS6 Commands
*
4EBD             DEFM    'APPEND'        Text for SYS6 cmd
4EC3             DEFW    4E90H           Addr where SYS6 is called
4EC6             DEFM    'ATTRIB'        Text for SYS6 cmd
4ECB             DEFW    4E90H           Addr where SYS6 is called
4ECD             DEFM    'AUTO  '        Text for SYS6 cmd
4ED3             DEFW    4E90H           Addr where SYS6 is called
4ED5             DEFM    'CLOCK '        Text for SYS6 cmd
4EDB             DEFW    4E90H           Addr where SYS6 is called
4EDD             DEFM    'COPY  '        Text for SYS6 cmd
4EE3             DEFW    4E90H           Addr where SYS6 is called
4EE5             DEFM    'DATE  '        Text for SYS6 cmd
4EEB             DEFW    4E90H           Addr where SYS6 is called
4EED             DEFM    'DEVICE'        Text for SYS6 cmd
4EF3             DEFW    4E90H           Addr where SYS6 is called
4EF5             DEFM    'DIR   '        Text for SYS6 cmd
4EFB             DEFW    4E90H           Addr where SYS6 is called
4EFD             DEFM    'DUMP  '        Text for SYS6 cmd
4F03             DEFW    4E90H           Addr where SYS6 is called
4F05             DEFM    'FREE  '        Text for SYS6 cmd
4F0B             DEFW    4E90H           Addr where SYS6 is called
4F0D             DEFM    'KILL  '        Text for SYS6 cmd
4F13             DEFW    4E90H           Addr where SYS6 is called
4F15             DEFM    'LIB   '        Text for SYS6
4F1B             DEFW    4E90H           Addr where SYS6 is called
4F1D             DEFM    'LIST  '        Text for SYS6
4F23             DEFW    4E90H           Addr where SYS6 is called
4F25             DEFM    'LOAD  '        Text for SYS6
4F2B             DEFW    4E90H           Addr where SYS6 is called
4F20             DEFM    'PRINT '        Text for SYS6
4F33             DEFW    4E90H           Addr where SYS6 is called
4F35             DEFM    'PROT  '        Text for SYS6
4F3B             DEFW    4E90H           Addr where SYS6 is called
4F3D             DEFM    'RENAME'        Text for SYS6
4F43             DEFW    4E90H           Addr where SYS6 is called
4F45             DEFM    'TIME  '        Text for SYS6
4F4B             DEFW    4E90H           Addr where SYS6 is called
4F4D             DEFM    'VERIFY'        Text for SYS6
4F35             DEFW    4E90H           Addr where SYS6 is called
4F55             DEFB    00              Text string terminator
*
* Begin processing for option 50. Copy string specified in
* HL to addr. specified in DE. Source string is assumed to
* be a file name. Dest. addr. is assumed to be a DCB. Source
* string is edited according to file name conventions e.g. no
* special characters etc. On exit, status register is zero if
* no errors were detected, otherwise it is non-zero.
*
* HL= String address
* B = No. chars. in string
* DE= DCB Address
*
4F56 D5          PUSH    DE              Save DCB address.
4F57 0608        LD      B,08H           No. of chars. to examine in string
4F59 CDF44F      CALL    4FF4H           Look for /,: or . in cmd string
4F5C 203A        JR      NZ,4F98H        Jump if no chars. copied to DCB
4F5E FE2F        CP      2FH             Was it terminated by an extension
4F60 2009        JR      NZ,4F6BH        Jmp, no extension specified
4F62 12          LD      (DE),A          Move / to DCB
4F63 13          INC     DE              Bump to next location in DCB
4F64 0603        LD      B,03H           No. of characters to move
4F66 CDF44F      CALL    4FF4H           Move extension to DCB
4F69 203A        JR      NZ,4FA5H        Jump if no extension
4F6B FE2E        CP      2EH             Was there a password present
4F6D 2009        JR      NZ,4F78H        Jmp, no password
4F6F 12          LD      (DE),A          Move . to DCB
4F70 13          INC     DE              Bump to next position in DCB
4F71 0608        LD      B,08H           B = No. of Characters in password
4F73 CDF44F      CALL    4FF4H           Move password to DCB
4F76 202D        JR      NZ,4FA5H        Jump if no password present
4F78 FE3A        CP      3AH             Was there a drive specification (:)
4F7A 2009        JR      NZ,4F85H        No Jump if no,
4F7C 12          LD      (DE),A          Yes. Move : to DCB
4F7D 13          INC     DE              Bump to next position
4F7E 0601        LD      B,01H           Max. number of chars. in drive spec.
4F80 CDF44F      CALL    4FF4H           Copy drive specification to DCB
4F83 2020        JR      NZ,4FA5H        Go if no drive number following :
4F85 4F          LD      C,A             Save terminating characters
4F86 3E03        LD      A,03H           Terminating character for DCB list
4F88 12          LD      (DE),A          To DCB
4F89 AF          XOR     A               Clear status flags
4F8A 79          LD      A,C             Restore term. char. to A
4F8B D1          POP     DE              Restore orgin. of DCB to DE
4F8C D5          PUSH    DE              And save it on stack
4F8D 01A74F      LD      BC,4FA7H        Addr. of TO, ON, OVER phrases
4F90 CD2A50      CALL    502AH           Look for match
4F93 D1          POP     DE              Restore DCB address
4F94 28C0        JR      Z,4F56H         Jump if TO/ON/OVER phrase found
4F96 AF          XOR     A               Signal good status and
4F97 C9          RET                     Return to caller
*
*
* Entered when no characters copied to DCB. If command begins
* with an *, and two characters follow, copy command to DCB.
*
*
4F98 FE2A        CP      2AH             Test for *
4F9A 2009        JR      NZ,4FA5H        Jump if not *
4F9C 12          LD      (DE),A          Move * to DCB
4F9D 13          INC     DE              Bump to next position in DCB
4F9E 0602        LD      B,02H           Number of characters to scan
4FA0 CDF44F      CALL    4FF4H           Copy next two characters to DCB
4FA3 28E0        JR      Z,4F85H         Jump if two were copied.
                                         Terminate name w/03
4FA5 D1          POP     DE              Restore caller's DE
4FA6 C9          RET                     Return to caller
```

```
*
*               Parameter Seperator List
*
4FA7            DEFM    'TO    '
4FAD            DEFW    0000H
4FAF            DEFM    'ON    '
4FB5            DEFW    0000H
4FB7            DEFM    'OVER  '
4FBD            DEFW    0000H
4FBF            DEFB    00H             List terminator
*
* Begin processing for option 50. Add /CMD suffix to file
* name in DCB. Skip to end of file name (find last char of
* file name before drive specification). Move all chars after
* file name down 4 positions (create a hole for /CMD suffix).
* Add CMD suffix, if a / is detected following file name do
* not add suffix
*
4FC0 D5         PUSH    DE              Save DCB addr.
4FC1 E5         PUSH    HL              Save appendage list
4FC2 EB         EX      DE,HL           DE = appendage list. HL = DCB addr
4FC3 23         INC     HL              Point to the 2nd char. in DCB
4FC4 0609       LD      B,09H           Max.# of char.to test.
4FC6 7E         LD      A,(HL)          .   Get a char.from DCB
4FC7 FE2F       CP      2FH             .   Test for / (extension)
4FC9 280D       JR      Z,4FD8H         .   Go if / found (don't add suffix)
4FCB 380E       JR      C,4FDBH         .   Or go if special char.
                                        .   found (do not add suffix)
4FCD FE3A       CP      3AH             .   Test for : (drive spec.)
4FCF 3804       JR      C,4FD5H         .   Jump if numeric found
4FD1 FE41       CP      41H             .   Test for letter
4FD3 3806       JR      C,4FDBH         .   Jump if not letter. End of
                                        .   file name in DCB (03) assumed
4FD5 23         INC     HL              .   We have a letter. Bump to next
4FD6 10EE       DJNZ    4FC6H           .   char. & loop till all examined
4FD8 E1         POP     HL              Restore appendage list addr.
4FD9 D1         POP     DE              Restore DCB addr.
4FDA C9         RET                     Return to caller w/o adding suffix
*
*
*
4FDB 010F00     LD      BC,000FH        BC = 15, max distance to end of DCB
4FDE 09         ADD     HL,BC           Compute addr. of last byte used
4FDF 54         LD      D,H             DE = last byte
4FE0 5D         LD      E,L             used in DCB
4FE1 13         INC     DE              Now, add 4 to last used
4FE2 13         INC     DE              addr. so we can move
4FE3 13         INC     DE              file name down to create
4FE4 13         INC     DE              A "hole" for the /CMD suffix
4FE5 03         INC     BC              BC = number of bytes to move down
4FE6 EDB8       LDDR                    Move data after file name down 4
4FE8 E1         POP     HL              HL = start addr. of CMD text
4FE9 23         INC     HL              Skip to
4FEA 23         INC     HL              end of list for LDDR instr.
4FEB 0E03       LD      C,03H           C = number of chars. to copy
4FED EDB8       LDDR                    Copy CMD text to DCB
4FEF 3E2F       LD      A,2FH           A = "/"
4FF1 12         LD      (DE),A          Separate file name & suffix with "/"
4FF2 D1         POP     DE              Restore start of DCB addr.
4FF3 C9         RET                     Return to caller
*
* Copy characters from list in DE to HL. Copy terminator when
* (B) chars, or 03, or 0D encountered in source list.
*
4FF4 78         LD      A,B             B = Number of characters
                                        to examine
4FF5 322650     LD      (5026H),A       Save no. of chars. to test
4FF8 04         INC     B               Adjust for DEC at 5013
4FF9 7E         LD      A,(HL)          .   Get next symbol from
                                        .   command string
4FFA FE03       CP      03H             .   Was it an ETX
4FFC 2823       JR      Z,5021H         .   Jump if yes
4FFE FE0D       CP      0DH             .   No, test for Cr
5000 281F       JR      Z,5021H         .   Jump if Cr
5002 23         INC     HL              .   Bump to next char. in
                                        .   command string
5003 FE30       CP      30H             .   Was previous char. a
                                        .   special char.
5005 381A       JR      C,5021H         .   Jump, if it's a spec. char.
5007 FE3A       CP      3AH             .   Test for alph., numeric, or
                                        .   special symbol
5009 3808       JR      C,5013H         .   Jump if numeric
500B FE41       CP      41H             .   Test for alpha or
                                        .   special symbol
500D 3812       JR      C,5021H         .   Jump if spec. symbol
500F FE5B       CP      5BH             .   Test for upper case
5011 300E       JR      NC,5021H        .   Jump if upper case
5013 05         DEC     B               .   Count one legitimate
                                        .   character found
5014 2808       JR      Z,501EH         .   Jump if all tested
5016 12         LD      (DE),A          .   Move to DCB in case
                                        .   it's a file name
5017 AF         XOR     A               .   Signal at least one
5018 322650     LD      (5026H),A       .   Character copied to DCB
501B 13         INC     DE              .   Bump to next addr. in DCB
501C 18DB       JR      4FF9H           .   Loop until end of input or
                                        .   specified # of char. copied
501E 04         INC     B               .   Adjust count for last char.
501F 18D8       JR      4FF9H           .   Continue scan until
                                        .   terminating char. found,
                                        .   or (B) chars copied
5021 4F         LD      C,A             Save terminating char.
5022 3E03       LD      A,03H           and signal
5024 12         LD      (DE),A          End command name in DCB with a 03
5025 3E00       LD      A,00H           Count of characters tested
5027 B7         OR      A               Set status flags for count
5028 79         LD      A,C             Terminating character to A-reg.
5029 C9         RET                     Return to caller
*
* Compare list pointed to by BC against list pointed to by DE
*
502A E5         PUSH    HL              Save input string addr.
502B 60         LD      H,B             HL = Addr. of command
502C 69         LD      L,C             Tables to search
502D 0E01       LD      C,01H           C = Number of lists searched
502F 1A         LD      A,(DE)          .   Fetch a char.from search list
5030 BE         CP      (HL)            .   Compare it to input string
5031 280F       JR      Z,5042H         .   Jump if first char. matches
5033 C5         PUSH    BC              .   .   No match. Go to next list
                                        .   .   Save  caller's BC
5034 010800     LD      BC,0008H        .   .   Add 8 to current list to
5037 09         ADD     HL,BC           .   .   get addr. of next list
5038 C1         POP     BC              .   .   Restore caller's BC
5039 0C         INC     C               .   .   Bump count of lists searched
503A 7E         LD      A,(HL)          .   .   Fetch 1st char. of next list
503B B7         OR      A               .   .   test for end of list
503C 20F1       JR      NZ,502FH        .   .   Loop, not end of list
503E E1         POP     HL              .   .   Restore caller's HL
503F F601       OR      01H             .   Signal no match
5041 C9         RET                     .   Return to caller
*
* First char. in list matches. Compare remainder of list.
*
5042 0605       LD      B,05H           .   No. of chars. left to match
5044 E5         PUSH    HL              .   Save input string addr.
5045 D5         PUSH    DE              .   Save search list addr. Save
                                        .   starting addr. of command string
5046 13         INC     DE              .   .   Bump to next char. of input list
5047 23         INC     HL              .   .   and companion list
5048 1A         LD      A,(DE)          .   .   Fetch a char. from input list
5049 FE03       CP      03H             .   .   Test for end of string
504B 2827       JR      Z,5074H         .   .   Go if end of input string
504D FE0D       CP      0DH             .   .   Else test for end of line
504F 2823       JR      Z,5074H         .   .   Jump if end of input string
5051 BE         CP      (HL)            .   .   Does char. match search list
5052 2010       JR      NZ,5064H        .   .   No, go test for ending byte
5054 10F0       DJNZ    5046H           .   .   Yes, loop to test next char.
                                        .   .   until all chars. tested
5056 D1         POP     DE              .   .   DE = 2nd char. of matching
                                        .   .   search list
5057 79         LD      A,C             .   .   A = No. of lists searched
5058 C1         POP     BC              .   .   Starting addr. of matching cmd
5059 210600     LD      HL,0006H        .   .   HL = Length of command
505C 09         ADD     HL,BC           .   .   HL = Ending addr. of command
505D 4F         LD      C,A             .   .   C = List number
505E 5E         LD      E,(HL)          .   .   E = LSB of command addr.
505F 23         INC     HL              .   .   Bump to MSB
5060 56         LD      D,(HL)          .   .   D = MSB of command addr.
5061 E1         POP     HL              .   .   HL = Starting addr. of
                                        .   .   input string
5062 AF         XOR     A               .   .   Signal good status
5063 C9         RET                     .   .   Return to caller
*
*
*
5064 FE30       CP      30H             .   .   Was it a special char.
                                        .   .   (e.g. / . - + - *)
5066 380C       JR      C,5074H         .   .   Jump if yes
5068 FE3A       CP      3AH             .   .   No, was it a digit 0 - 9
506A 380D       JR      C,5079H         .   .   Jump if yes
506C FE41       CP      41H             .   .   No, was it a special char.
                                        .   .   (e.g : ; < = > ? @)
506E 3804       JR      C,5074H         .   .   Jump if yes
5070 FE5B       CP      5BH             .   .   No, was it alpha A - Z
5072 3805       JR      C,5079H         .   .   Jump if yes
5074 7E         LD      A,(HL)          .   .   No, fetch current char.
5075 FE20       CP      20H             .   .   Is it a space
5077 28DD       JR      Z,5056H         .   .   Jump, if yes
5079 D1         POP     DE              .   No, clear stack
507A E1         POP     HL              .   Restore input string addr.
507B 18B6       JR      5033H           Get next list addr. and retest
*
*       ---  Begin processing for option 60 parse  ---
*
507D 7E         LD      A,(HL)          .   Get a character
507E FE0D       CP      0DH             .   Test for end of line (C.R.)
5080 C8         RET     Z               .   Exit if a "CR" found
5081 FE28       CP      28H             .   Test for left parenthesis
5083 2809       JR      Z,508EH         .   Jump if "("
5085 FE20       CP      20H             .   Test for space
5087 2802       JR      Z,508BH         .   Jump if space
5089 B7         OR      A               .   Set status flags
508A C9         RET                     .   Return to caller
508B 23         INC     HL              .   Bump to next char.
508C 18EF       JR      507DH           .   Go test again
508E D5         PUSH    DE              .   Save caller's cmd list addr
508F 0606       LD      B,06H           .   B = Number of chars. to copy
5091 11AF51     LD      DE,51AFH        .   Buffer addr.
5094 23         INC     HL              .   Bump to first char. after "("
5095 CDF44F     CALL    4FF4H           .   Copy 6 chars or untill ")" found
5098 2B         DEC     HL              .   Backspace over terminating 03
5099 D1         POP     DE              .   Restore caller's DE
509A C0         RET     NZ              .   Exit if no chars. copied (error)
509B D5         PUSH    DE              .   Save caller's DE
509C 42         LD      B,D             .   Caller's command list table
509D 4B         LD      C,E             .   Addr. to BC for routine at 502A
509E 11AF51     LD      DE,51AFH        .   Parsed command line
50A1 CD2A50     CALL    502AH           .   Look for cmd in caller's cmd list
50A4 2802       JR      Z,50A8H         .   Jump if a match found
50A6 D1         POP     DE              .   .   Restore caller's DE
50A7 C9         RET                     .   .   Return with error status
*
*
```

```
50A8 7E         LD      A,(HL)          . .   Refetch char. from input string
50A9 FE3D       CP      3DH             . .   Compare it to an =
50AB 2814       JR      Z,50C1H         . .   Jump if =
50AD 01FFFF     LD      BC,0FFFFH       . .   Value returned if no = parameter
50B0 79         LD      A,C             . . .   Save LSB of addr.
                                        . . .   or hex value
50B1 12         LD      (DE),A          . . .   In buffer
50B2 13         INC     DE              . . .   Bump to next loc.
50B3 78         LD      A,B             . . .   Get MSB of addr.
50B4 12         LD      (DE),A          . . .   and save in buffer
50B5 D1         POP     DE              . . .   Restore caller's
                                        . . .   command list address
50B6 7E         LD      A,(HL)          . . .   Refetch Char. from
                                        . . .   input string
50B7 FE2C       CP      2CH             . . .   Compare with '
50B9 28D3       JR      Z,50BEH         . . .   Jump if '
50BB FE29       CP      29H             . .   Not ', compare it to a )
50BD C0         RET     NZ              . .   Exit if not a )
50BE 23         INC     HL              . .   Bump to next char.
50BF AF         XOR     A               . .   Signal good status
50C0 C9         RET                     . .   Return to caller
50C1 23         INC     HL              . .   Bump to first char.
                                        . .   after =
50C2 7E         LD      A,(HL)          . .   Fetch char.
50C3 FE58       CP      58H             . .   Test for X (hex flag
50C5 280B       JR      Z,50D2H         . .   Go convert if hex value
50C7 FE41       CP      41H             . .   Test for digit
50C9 380F       JR      C,50DAH         . .   Jump if digit
50CB CDDF50     CALL    50DFH           . .   Alpha, go test for N, Y, OF
50CE 28E0       JR      Z,50B0H         . .   Jump if N, Y, or OF found
50D0 18D4       JR      50A6H           . .   Error, return to caller
50D2 23         INC     HL              . .   Bump to next char.
50D3 CD1F51     CALL    511FH           . .   Convert ASCII hex to binary
50D6 28D8       JR      Z,50B0H         . .   If no error save hex value
50D8 18CC       JR      50A6H           . .   Else, return to caller
50DA CD0451     CALL    5104H           .   Go get decimal value
50DD 18D1       JR      50B0H           .   and return to caller
*
*           --- Test for YES, NO or OFF parameter ---
*
50DF 010000     LD      BC,0000H        Signal N,Y,or OF
50E2 7E         LD      A,(HL)          Fetch first symbol past "(" or =
50E3 FE59       CP      59H             Test for Y
50E5 2810       JR      Z,50F7H         Jump if Y
50E7 FE4E       CP      4EH             No, test for N
50E9 280F       JR      Z,50FAH         Jump if N
50EB FE4F       CP      4FH             Not Y/N, test for O
50ED C0         RET     NZ              Return if not O
50EE 23         INC     HL              Char. is O, bump to next char
50EF 7E         LD      A,(HL)          Fetch char. after O
50F0 FE46       CP      46H             And test for F
50F2 2806       JR      Z,50FAH         Jump if OF found
50F4 FE4E       CP      4EH             Not F, test for N
50F6 C0         RET     NZ              Return if ON
50F7 01FFFF     LD      BC,0FFFFH       Signal ON
50FA 23         INC     HL              .   Bump to next input string char.
50FB 7E         LD      A,(HL)          .   Fetch next char.
50FC FE29       CP      29H             .   Test for ")"
50FE C8         RET     Z               .   Return if )
50FF FE2C       CP      2CH             .   Test for ","
5101 C8         RET     Z               .   Return if ,
5102 18F6       JR      50FAH           .   Loop until ) or , found
*
*          ---  ASCII to binary conversion routine  ---
*
5104 010000     LD      BC,0000H        Clear accumulator
5107 7E         LD      A,(HL)          .   Fetch an ASCII character
5108 D630       SUB     30H             .   Get its decimal equivalent
510A D8         RET     C               .   Exit if not a digit
510B FE0A       CP      0AH             .   Test for value greater than 9
510D D0         RET     NC              .   Exit if char. greater than 9
510E E5         PUSH    HL              .   Save input string addr.
510F 60         LD      H,B             .   HL = Accumulated
5110 69         LD      L,C             .   Value, thus far
5111 29         ADD     HL,HL           .   Value, * 2
5112 29         ADD     HL,HL           .   Value, * 4
5113 09         ADD     HL,BC           .   Value, * 5
5114 29         ADD     HL,HL           .   Value, * 10
5115 0600       LD      B,00H           .   Prohibit overflow during add
5117 4F         LD      C,A             .   BC = Current digit
5118 09         ADD     HL,BC           .   Add it to previous value
5119 44         LD      B,H             .   Then, move current
511A 4D         LD      C,L             .   value to BC
511B E1         POP     HL              .   Restore cmd string address
511C 23         INC     HL              .   Bump to next char.
511D 18E8       JR      5107H           .   Loop till end of line detected
*
*     Scan input string following ( or = look for Y, N, or OF. If
*     found, return W/BC = 0 after skipping to next , or ) in cmd
*     string. Look for ON. If found, return W/BC = -1 and
*     position to next , or ) in cmd string. If Y, N, or O does
*     not follow as next char, exit W/BC = 0.
*
511F 010000     LD      BC,0000H        Zero accumulator
5122 7E         LD      A,(HL)          Get a char. from cmd string
5123 FE27       CP      27H             Compare it to an ' (apostraphe)
5125 C0         RET     NZ              Exit if value not preceded by ' (error)
5126 23         INC     HL              Bump to first digit
5127 7E         LD      A,(HL)          .   Fetch a digit
5128 D630       SUB     30H             .   Convert it to binary
512A 380A       JR      C,5036H         .   Jump if end of input string
512C FE0A       CP      0AH             .   Test for digit 0-9
512E 380E       JR      C,513EH         .   Jump if 0-9
5130 D607       SUB     07H             .   Test for lettdr A-F
5132 FE10       CP      10H             .   And adjust binary value
5134 3808       JR      C,513EH         .   Jump if A - F
5136 7E         LD      A,(HL)          .   Refetch non-digit char.
5137 FE27       CP      27H             .   Test for end of hex digit (')
5139 23         INC     HL              .   Bump to next input string char.
513A C8         RET     Z               .   Exit if done
513B 2B         DEC     HL              .   Else back-up to illegal char.
513C AF         XOR     A               .   Signal an error
513D C9         RET                     .   and return to caller
*
*       Multiply current digit by 16 and add to accumulator.
*
513E E5         PUSH    HL              .   Save input string addr.
513F 60         LD      H,B             .   Move current hex binary value
5140 69         LD      L,C             .   To HL
5141 29         ADD     HL,HL           .   Multiply current hex value by
5142 29         ADD     HL,HL           .   16 using additions
5143 29         ADD     HL,HL           .   *8
5144 29         ADD     HL,HL           .   *16
5145 44         LD      B,H             .   Save MSB of new hex value
5146 85         ADD     A,L             .   Add current hex digit
5147 0F         LD      C,A             .   BC = Current hex value
5148 E1         POP     HL              .   Restore input string addr.
5549 23         INC     HL              .   Bump to next digit
514A 18DB       JR      5127H           .   Loop till non-hex found
*
*           BASIC 2 Command Processing
*
514C 21D206     LD      HL,06D2H        Addr. of jump vectors
514F 110040     LD      DE,4000H        And keyboard/video/and LP
5152 013600     LD      BC,0036H        DCB's in ROM
5155 EDB0       LDIR                    Move jump vectors & DCB's to RAM
5157 AF         XOR     A               Load a zero in A-reg., so we can zero
5158 0627       LD      B,27H           the 39 bytes following the DCB's
515A 12         LD      (DE),A          .   Store a zero
515B 13         INC     DE              .   Bump store address
515C 10FC       DJNZ    515AH           .   Loop until 39 bytes zeroed
515E F3         DI                      Kill interrupts
515F C37500     JP      0075H           Go to LEVEL II ROM warmstart
                                        (enter ROM BASIC)
*
*           DEBUG Command Processing
*
5162 CDA051     CALL    51A0H           Process (ON/OFF) part of DEBUG cmd.
5165 210F43     LD      HL,430FH        Addr. of system condition flag
5168 F3         DI                      Kill interrupts. Flag fussing
                                        with 4015/4016
5169 2819       JR      Z,5184H         Jump if DEBUG, OFF
516B CBFE       SET     07H,(HL)        Else, signal DEBUG, ON
516D 210F40     LD      HL,400FH        Addr. of load DEBUG vector
5170 3EC3       LD      A,0C3H          JP OP Code
5172 221643     LD      (4316H),HL      Construct a JP 400F instr. and
5175 321543     LD      (4315H),A       store at 4315, this yields a
                                        JP 44B4 (load DEBUG call)
5178 3ECD       LD      A,0CDH          Call OP Code
517A 223140     LD      (4031H),HL      Save addr. 400F at 4031 - 4032
517D FB         EI                      Enable interrupts
517E 323040     LD      (4030H),A       then save a CALL at 4030 giving
                                        a 4030: CALL 400FH
5181 C31F4E     JP      4E1FH           Go read next command
*
*           DEBUG, OFF Processing
*
5184 CBBE       RES     07H,(HL)        Clear DEBUG flag in
                                        system condition byte
5186 AF         XOR     A               then clear
5187 321543     LD      (4315H),A       active flag and restore 4030-4032
518A 3E3E       LD      A,3EH           LD  A, A3  OP Code
518C 21A3EF     LD      HL,0EFA3H       A3 for LD A followed by a RST 28
518F 18E9       JR      517AH           Store A, A3/RST 28 at
                                        4030/4031/4032
*
*           TRACE Command Processing
*
5191 CDA051     CALL    51A0H           Get (ON/OFF)
5194 11D94C     LD      DE,4CD9H        Addr. of TRACE off,
                                        routine in SYS 0
5197 3E0B       LD      A,0BH           Index of TRACE in SYS0
                                        clock scan list
5199 CA1344     JP      Z,4413H         Jump if TRACE (ON). Add TRACE to
                                        clock scan list
519C C31044     JP      4410H           Else, assume TRACE (OFF).
                                        Remove TRACE from clock scan list
*
* Get next char. from input string. If (, test for YES, NO or OFF.
*
519F 23         INC     HL              .   Bump to next symbol in cmd string
51A0 7E         LD      A,(HL)          .   Get next symbol from cmd string
51A1 FE20       CP      20H             .   Compare it with a blank
51A3 28FA       JR      Z,519FH         .   If a blank, fetch and
                                        .   test next symbol.
                                        .   Loop until non-blank clear
51A5 FE28       CP      28H             Not a blank, is it a left paren.
51A7 C0         RET     NZ              Return to caller if not (
51A8 23         INC     HL              Bump past starting parenthesis.
51A9 CDDF50     CALL    50DFH           Look for Y, N, OF or ON phrase
51AC 78         LD      A,B             A = 0 if Y,N, or OF. A = FF if ON
51AD B7         OR      A               Set status flags for Y, N, F, ON
51AE C9         RET                     Return to caller
*
*
*
51AF                    DS 6            local buffer for holding parsed cmd.
*
*
*
51B5            DEFM    'CMD'           Suffix for file loads
51B8            DEFB    00B             Terminatior
51B9            DEFM    'DOS READY'     Command prompt message
51C2            DEFB    0DH             Terminator
51C3            DEFM    'WHAT'
51C8            DEFB    0DH             Terminator
```

```
*************************************************************
*                                                           *
*                        SYS2/SYS                           *
*                  OPEN and INIT Processing                 *
*                                                           *
*************************************************************


4E00 E670       AND     70H             Isolate option
4E02 FE10       CP      10H             Test for OPEN request
4E04 CA124E     JP      Z,4E12H         Jump if OPEN
4E07 FE20       CP      20H             Test for INIT request
4E09 CAD84E     JP      Z,4ED8H         Jump if INIT
4E0C FE30       CP      30H             Test for create overflow entry
4E0E CA504F     JP      Z,4F50H         Jump if create overflow entry
4E11 C9         RET                     None of the above - Ret caller
*
*                       Begin OPEN Processing
*
4E12 CD9648     CALL    4896H           Save regs. setup return sequence
4E15 78         LD      A,B             A = Record length
4E16 32AF4F     LD      (4FAFH),A       Save record length
4E19 22C04F     LD      (4FC0H),HL      Callers sector buffer address
4E1C DDE5       PUSH    IX              DCB addr. to HL via stack
4E1E E1         POP     HL              HL = DCB addr.
4E1F CD2750     CALL    5027H           Move file name to local DCB (515D)
4E22 C0         RET     NZ              Exit if illegal chars. found
4E23 215D51     LD      HL,515DH        HL = Start of local DCB
4E26 CD9B50     CALL    509BH           Compute hash code for filename
4E29 324E4E     LD      (4E4EH),A       Save hash code
4E2C 115551     LD      DE,5155H        Addr. of password buffer
4E2F CDD150     CALL    50D1H           Get encode of password into HL
4E32 226851     LD      (5168H),HL      And save in
4E35 226A51     LD      (516AH),HL      Local DCB
4E38 3A5451     LD      A,(5154H)       Get drive specification flag
4E3B 4F         LD      C,A             C = Default unit number (0)
4E3C 3C         INC     A               = FF if none given, else = drive no.
4E3D 2001       JR      NZ,4E40H        Jump if drive specified
4E3F 4F         LD      C,A             Else begin with drive 0, and
4E40 CDFD50     CALL    50FDH             .   see if drive in C reg. present
4E43 200F       JR      NZ,4E54H          .   Jump if drive not present
4E45 CD2E51     CALL    512EH             .   Drive found, get table in 4D00
4E48 C0         RET     NZ                .   Exit if error during read
4E49 7E         LD      A,(HL)            .   Get hash code from HIT
                                          .   just read
4E4A B7         OR      A                 .   Set status flags for hash code
4E4B 2804       JR      Z,4E51H           .   Jump if slot empty
4E4D FE00       CP      00H               .   Compare hash from HIT
                                          .   to computed
4E4F 2819       JR      Z,4E6AH           .   Jump if hash codes match
4E51 2C         INC     L                 .   Else bump to next loc.
                                          .   in hash tab.
4E52 20F5       JR      NZ,4E49H          .   Jump if hash tab. not finished
4E54 3A5451     LD      A,(5154H)         .   No match. File not found,
                                          .   search next drive
4E57 3C         INC     A                 .   Unless drive was specified
4E58 2006       JR      NZ,4E60H          .   If drive specified
                                          .   file not found
4E5A 0C         INC     C                 .   Else bump drive number
4E5B 79         LD      A,C               .   Drive to A-reg. so we can test
4E5C FE04       CP      04H               .   if all drives were searched
4E5E 38E0       JR      C,4E40H           .   Jump if all drives
                                          .   not searched
4E60 3E18       LD      A,18H           Error code for file not found
4E62 B7         OR      A               Set status flags
4E63 C9         RET                     Return to caller
*
*               Clear Stack and Prepare to Search HIT
*
4E64 C1         POP     BC              Clear stack
4E65 E1         POP     HL              Clear stack
4E66 C1         POP     BC              Restore drive no. to C-reg.
4E67 E1         POP     HL              Restore current hash sector index
4E68 18E7       JR      4E51H           Go test next hash value
*
*             Read Directory Sector Containing File Entry
*                     Matching Hash Code Found
*
4E6A E5         PUSH    HL              Save hash code address
4E6B C5         PUSH    BC              Save B/unit number
4E6C 45         LD      B,L             B = Addr. of entry in directory
4E6D CDC14A     CALL    4AC1H           Read directory sector into 4200
4E70 2803       JR      Z,4E75H         Continue if no error
4E72 C1         POP     BC              Else clear
4E73 E1         POP     HL              Stack and
4E74 C9         RET                     Return to caller w/error status
*
*          ---  Locate File Entry io Directory Sector  ---
*
4E75 E5         PUSH    HL              Save addr. of file entry
4E76 C5         PUSH    BC              And dir index, unit number
4E77 CB7E       BIT     07H,(HL)        Test if primary directory entry
4E79 20E9       JR      NZ,4E64H        Jump if not primary entry
4E7B 3E05       LD      A,05H           Offset to name in dir
4E7D 85         ADD     A,L             Add 5 to addr. of entry
4E7E 6F         LD      L,A             HL = Addr. of name in directory
4E7F 115D51     LD      D ,515DH        DE = Addr. of local DCB
                                        (contains name)
4E82 060B       LD      B,0BH           Max No. of chars. to match
4E84 1A         LD      A,(DE)          A = Char. from local DCB
4E85 BE         CP      (HL)            Compare to Char. in directory sector
4E86 20DC       JR      NZ,4E64H        Go to next entry if no match, else
4E88 23         INC     HL              Bump directory entry address and
4E89 13         INC     DE              Local DCB address
4E8A 10F8       DJNZ    4E84H           Loop till all chars. in name matched
*
*                 Filename Located in Directory Sector
*           Isolate Access Permissions and Password Encodes
*
4E8C C1         POP     BC              Restore unit number
4E8D 79         LD      A,C             Move unit number to A reg.
4E8E 325451     LD      (5154H),A       Save unit number
4E91 E1         POP     HL              HL = directory entry sector addr.
4E92 F1         POP     AF              Clear stack
4E93 F1         POP     AF              Clear stack
4E94 C5         PUSH    BC              Save index/unit numcer
4E95 E5         PUSH    HL              Addr. of directory entry
                                        containing file
4E96 7E         LD      A,(HL)          Get first byte of directory entry
4E97 E607       AND     07H             Isolate permission bits
4E99 4F         LD      C,A             And save in C
4E9A 0600       LD      B,00H           Permission flags - no restriction
4E9C 3E10       LD      A,10H           Offset to password
4E9E 85         ADD     A,L             Gives addr. of update password
4E9F 6F         LD      L,A             HL = Addr. of password in dir entry
4EA0 ED5B6A51   LD      DE,(516AH)      Encode of password
4EA4 E5         PUSH    HL              Save addr. of dir (16) entry
4EA5 21A261     LD      HL,61A2H        HL = 24,994
4EA8 AF         XOR     A               Clear carriage
4EA9 ED52       SBC     HL,DE           24,994 - encode of password
4EAB E1         POP     HL              Restore addr. of password
4EAC 1821       JR      4ECFH           Go fill in DCB.
*
*
*
4ECF E1         POP     HL              HL = Addr.of directory entry
4ED0 78         LD      A,B             A = Access permission
4ED1 32B24F     LD      (4FB2H),A       Save for storage into DCB
4ED4 C1         POP     BC              Restore drive number to C-reg.
4ED5 C3A74F     JP      4FA7H           Go fill in DCB
*
*                          INIT Processing
*
4ED8 CD9648     CALL    4896H           Save regs, setup exit sequence
4EDB CD154E     CALL    4E15H           Make OPEN call
4EDE C8         RET     Z               Return if file found
4EDF FE18       CP      18H             Test for file not found. Fall
                                        through if file not there
4EE1 C0         RET     NZ              Return if some other error
*
*             ---  File Does not Exist. Create Entry  ---
*
4EE2 3E10       LD      A,10H           Byte 0 of dir to be assigned
4EE4 321D4F     LD      (4F1DH),A       Entry assigned. Unrestricted access
4EE7 3A5451     LD      A,(5154H)       Get drive specification
4EEA 4F         LD      C,A             Save it in C-reg.
4EEB 3C         INC     A               Test if drive no. was specified
4EEC 2001       JR      NZ,4EEFH        Jump if unit specified
4EEE 4F         LD      C,A             Current unit number to C-reg.
4EEF CDFD50     CALL    50FDH           .   Go retest unit
4EF2 200B       JR      NZ,4EFFH        .   Jump if drive not present
4EF4 3809       JR      C,4EFFH         .   Jump if drive not ready
4EF6 CD2E51     CALL    512EH           .   Read hash table
4EF9 C0         RET     NZ              .   Exit if error during HIT read
4EFA CDAB50     CALL    50ABH           .   Find available slot in HIT
4EFD 280F       JR      Z,4F0EH         .   Go if slot for hash code found
4EFF 3A5451     LD      A,(5154H)       .   None left. Get drive spec.
4F02 3C         INC     A               .   Was a drive specified
4F03 2006       JR      NZ,4F0BH        .   Jump if yes (directory is full)
4F05 0C         INC     C               .   Bump to next drive
4F06 79         LD      A,C             .   So we can test drive number
4F07 FE04       CP      04H             .   Have all drives been searched
4F09 38E4       JR      C,4EEFH         .   If not, loop - else return
4F0B 3E1A       LD      A,1AH           space full error status
4F0D C9         RET                     Return to caller
4F0E 45         LD      B,L             B = Directory sector no
4F0F 3A4E4E     LD      A,(4E4EH)       Get hash code
4F12 77         LD      (HL),A          Save in HIT
4F13 CD4151     CALL    5141H           Write updated HIT
4F16 C0         RET     NZ              Ret. if error during write
4F17 CDC14A     CALL    4AC1H           Read directory sector that will
                                        contain file entry
4F1A C0         RET     NZ              Exit if error during read
4F1B E5         PUSH    HL              Save starting addr. of dir entry
4F1C 3600       LD      (HL),00H        Clear access control
4F1E 23         INC     HL              Bump to byte 2
4F1F 3600       LD      (HL),00H        Clear overflow pointer
4F21 23         INC     HL              Bump to byte 3
4F22 3600       LD      (HL),00H        Clear reserved byte
4F24 23         INC     HL              Bump to Byte 4
4F25 3600       LD      (HL),00H        Clear EOF offset byte
4F27 23         INC     HL              Bump to 5th byte
4F28 3AAF4F     LD      A,(4FAFH)       Get record length
4F2B 77         LD      (HL),A          And save in directory entry
4F2C 23         INC     HL              Bump to 6th byte in directory
4F2D C5         PUSH    BC              Preserve BC.
4F2E 010F00     LD      BC,000FH
4F31 EB         EX      DE,HL           DE = dir addr. for name
4F32 215D51     LD      HL,515DH        Internal DCB addr.
4F35 EDB0       LDIR                    Copy file name to dir entry
4F37 EB         EX      DE,HL           HL = Byte 15 of directory sector
4F38 3600       LD      (HL),00H        Clear EOF sector number (LSB)
4F3A 23         INC     HL              Bump to 16th byte of dir sector
4F3B 3600       LD      (HL),00H        Clear EOF sector number (MSB)
4F3D 23         INC     HL              Bump to 17th byte of directory
4F3E 060A       LD      B,0AH           Number of bytes to initialize
4F40 36FF       LD      (HL),0FFH       .   Store an FF in GAP field of
4F42 23         INC     HL              .    dir entry. Bump to next entry
4F43 10FB       DJNZ    4F40H           .   loop till all 10 GAPs initialized
4F45 C1         POP     BC              Restore HIT slot index used to
                                        derive dir. sector #.
4F46 CDD64A     CALL    4AD6H           Write updated directory sector
4F49 E1         POP     HL              Restore directory entry addr. to HL
```

```
4F4A C0          RET     NZ              Return if error
4F4B CDA74F      CALL    4FA7H           Go initialize DCB in user's area
4F4E 37          SCF                     Signal file not previously opened
4F4F C9          RET                     Return to caller
*
*                  Construct an Overflow Directory Entry
*
*       1. Find an available slot in HIT sector
*       2. Move file hash code to HIT slot located above
*       3. Write updated HIT
*       4. Read directory sector for HIT slot assigned
*       5. Initialize directory sector entry for HIT assigned
*          in step 1, as an overflow entry
*       6. Write directory sector containing overflow entry
*       7. Read directory sector containing primary entry
*       8. Update primary entry with overflow flag and pointer
*       9. Write updated primary entry
*
4F50 DD4E06      LD      C,(IX+06H)      C = drive number
4F53 CD2E51      CALL    512EH           Read HIT
4F56 C0          RET     NZ              Return if error during read
4F57 DD7E07      LD      A,(IX+07H)      A = Dir sector offset for this file
4F5A CDB650      CALL    50B6H           Find an available hole in HIT
4F5D 3E1E        LD      A,1EH           Error code for directory full
4F5F C0          RET     NZ              Return w/error if hash table full
4F60 45          LD      B,L             B = Next avail. loc. in HIT
4F61 7D          LD      A,L             Move it to A-reg.
4F62 32A24F      LD      (4FA2H),A       Save address of overflow entry
4F65 54          LD      D,H             D = MSB addr. of HIT buffer
4F66 DD5E07      LD      E,(IX+07H)      E = LSB addr. of HIT buffer
                                         (index for this file)
4F69 1A          LD      A,(DE)          Get hash code for file from HIT
4F6A 77          LD      (HL),A          Copy it to overflow entry
4F6B CD4151      CALL    5141H           Write updated HIT
4F6E C0          RET     NZ              Return if error during WRITE
4F6F CDC14A      CALL    4AC1H           Read dir sector for HIT slot assigned
4F72 C0          RET     NZ              Ret if directory read error
4F73 3690        LD      (HL),90H        Flag dir entry as assigned/overflow
4F75 2C          INC     L               Bump to 2nd byte of dir. entry
4F76 C5          PUSH    BC              Preserve unit number (C-reg.)
4F77 3ABE4A      LD      A,(4ABEH)       A = dir sector index for this file
4F7A 77          LD      (HL),A          Save as 2nd byte of overflow entry
4F7B 2C          INC     L               HL = 3rd byte of overflow entry
4F7C 0614        LD      B,14H           B = no. of bytes to zero
4F7E 3600        LD      (HL),00H        Zero a byte
4F80 2C          INC     L               Bump to next byte of overflow entry
4F81 10FB        DJNZ    4F7EH           Loop till 20 bytes cleared
4F83 E5          PUSH    HL              Save addr. of 1st GAP in entry
4F84 060A        LD      B,0AH           Number of bytes to initialize
4F86 36FF        LD      (HL),0FFH       Initialize
4F88 2C          INC     L               Bump to next byte
4F89 10FB        DJNZ    4F86H           Loop till all GAP's initialized
4F8B D1          POP     DE              DE = addr. of 1st GAP in entry
4F8C 13          INC     DE              Bump to number of grans (17th byte)
4F8D C1          POP     BC              Restore unit number to C-reg.
4F8E CDD64A      CALL    4AD6H           Write updated directory sector
                                         (contains overflow entry)
4F91 C0          RET     NZ              Exit if error during write
4F92 3ABE4A      LD      A,(4ABEH)       Get sector number for primary entry
4F95 47          LD      B,A             Move to B for nucleus read routine
4F96 CDC14A      CALL    4AC1H           Read primary entry sector
4F99 C0          RET     NZ              Exit if error during read
4F9A 7D          LD      A,L             HL = Addr of primary entry in sector
4F9B C61E        ADD     A,1EH           Position to 1st byte of 5th GAP
4F9D 6F          LD      L,A             HL = Directory entry (IE)
4F9E 36FE        LD      (HL),0FEH       Signal an overflow entry exists
4FA0 2C          INC     L               Bump to 2nd byte of 5th GAP
4FA1 3600        LD      (HL),00H        Save pointer to overflow entry
4FA3 CDD64A      CALL    4AD6H           Write updated primary entry
4FA6 C9          RET                     Return to caller
*
*      Fill out DCB. Copy portions of primary entry to DCB. On
*       entry DE = addr of directory entry, HL = addr of DCB.
*
4FA7 EB          EX      DE,HL           DE addr. of directory entry
4FA8 DDE5        PUSH    IX              Prepare to load HL
4FAA E1          POP     HL              HL = addr. of caller's DCB
4FAB 3680        LD      (HL),80H        Signal file open
4FAD 23          INC     HL              Bump to 2nd byte of DCB
4FAE 3E00        LD      A,00H           A = record length
4FB0 B7          OR      A               Set status flags for record length
4FB1 3E00        LD      A,00H           Load access flags (put here by 4ED1)
4FB3 2802        JR      Z,4FB7H         Jump if rec. length is
                                         zero (phys. I/O)
4FB5 F680        OR      80H             Else, signal logical I/O
4FB7 F620        OR      20H             Force read on first access
4FB9 77          LD      (HL),A          Save access flags, type of I/O
4FBA 23          INC     HL              Bump to 3rd byte of DCB
4FBB 3600        LD      (HL),00H        and store a zero (Not used)
4FBD 23          INC     HL              Bump to 4th byte of DCB
4FBE D5          PUSH    DE              Save original DCB addr.
4FBF 110000      LD      DE,0000H        Sector buffer addr. to DE
4FC2 73          LD      (HL),E          LSB of buffer addr. to DCB
4FC3 23          INC     HL              Bump to 5th byte of DCB
4FC4 72          LD      (HL),D          MSB of buffer to DCB
4FC5 23          INC     HL              Bump to 6th byte of DCB
4FC6 D1          POP     DE              Restore dir. entry addr.
4FC7 3600        LD      (HL),00H        Offset to end of cusrent record
4FC9 23          INC     HL              Bump to 7th byte of DCB
4FCA 71          LD      (HL),C          Save drive no.
4FCB 23          INC     HL              Bump to 8th byte of DCB
4FCC 70          LD      (HL),B          Save overflow pointer
4FCD 23          INC     HL              Bump to 9th byte of DCB
4FCE 3E03        LD      A,03H           Index to EOF byte in directory
4FD0 83          ADD     A,E             Add to LSB of addr. of
4FD1 5F          LD      E,A             base of directory sector
4FD2 1A          LD      A,(DE)          Load EOF byte offset from dir entry
4FD3 77          LD      (HL),A          Save EOF byte offset
4FD4 23          INC     HL              Bump to 10th byte of DCB
4FD5 13          INC     DE              Bump to next by of directory
4FD6 3AAF4F      LD      A,(4FAFH)       Load record length
4FD9 77          LD      (HL),A          Move record length to DCB
4FDA 23          INC     HL              Bump to 11th byte of DCB
4FDB 3600        LD      (HL),00H        Clear LSB of next record no.
4FDD 23          INC     HL              Bump to MSB
4FDE 3600        LD      (HL),00H        Clear MSB of next record no.
4FE0 23          INC     HL              Bump to 13th byte of DCB
4FE1 3E10        LD      A,10H           Index to EOF sector number
4FE3 83          ADD     A,E             Add to base of dir sector addr.
4FE4 5F          LD      E,A             Form addr. of 16th byte in DE
4FE5 1A          LD      A,(DE)          Fetch EOF sector (LSB)
4FE6 77          LD      (HL),A          And save in DCB
4FE7 23          INC     HL              Bump to 14th byte in DCB
4FE8 CD8051      CALL    5180H           Copy MSB of number of sectors
4FEB 23          INC     HL              Bump to 1st track addr. in DCB
4FEC 13          INC     DE              Bump to 1st trk. addr. in directory
4FED 1A          LD      A,(DE)          Fetch starting track number
4FEE 77          LD      (HL),A          Save in DCB
4FEF 23          INC     HL              Bump to gran count for 1st GAP in
4FF0 13          INC     DE              Dir sector and LSB of accumulated
                                         granule in DCB
4FF1 1A          LD      A,(DE)          Fetch # of granules from dir entry
4FF2 77          LD      (HL),A          Copy it to DCB
4FF3 23          INC     HL              Bump to track # for 2nd GAP
4FF4 E61F        AND     1FH             Isolate no.of consecutive
                                         granules -1 for 1st GAP
4FF6 3C          INC     A               Gives no. of consecutive grans
4FF7 4F          LD      C,A             BC will contain total grans
4FF8 0600        LD      B,00H
4FFA 3E04        LD      A,04H           No. of extant pairs remaining
4FFC F5          PUSH    AF              .   Save count.
4FFD 13          INC     DE              .   Bump to next GAP
4FFE 1A          LD      A,(DE)          .   Fetch next GAP from directory
4FFF FEFE        CP      0FEH            .   Look for end of GAP or
                                         .   overflow pointer
5001 3019        JR      NC,501CH        .   Jump if end or overflow pointer
5003 71          LD      (HL),C          .   Save LSB of total grans for
5004 23          INC     HL              .   previous GAP
5005 70          LD      (HL),B          .   Save MSB of total
5006223          INC     HL              .   Bump to first pos. in next GAP
5007 1A          LD      A,(DE)          .   Fetch track for next GAP
5008 77          LD      (HL),A          .   And save in DCB
5009 23          INC     HL              .   Bump to 2nd byte of GAP in DCB
500A 13          INC     DE              .   Bump to count in dir. sector
500B 1A          LD      A,(DE)          .   Get count of assigned grans
500C 77          LD      (HL),A          .   Save in DCB
500D 23          INC     HL              .   Bump to LSB of totals in DCB
500E E61F        AND     1FH             .   Isolate count from starting
                                         .   sector number
5010 3C          INC     A               .   Get true granule count
5011 81          ADD     A,C             .   Form new total granules
5012 4F          LD      C,A             .   Accessible thru
5013 78          LD      A,B             .   GAPS thus far
5014 CE00        ADC     A,00H           .   capture carry in of overflow
5016 47          LD      B,A             .   BC = Total granules accessible
5017 F1          POP     AF              .   Restore no. of GAP's to copy
5018 3D          DEC     A               .   Have we copied all GAP's
5019 20E1        JR      NZ,4FFCH        .   Jump if not
501B C9          RET                     Else ret to OPEN/INIT caller
*
*       End of GAP's in directory entry have been reached.
*       Fill remainder of DCB GAP area with bytes of FF.
*
501C F1          POP     AF              Number of unfilled entries
501D 07          RLCA                    Times four
501E 07          RLCA                    gives byte count
501F 47          LD      B,A             Move to B reg.for loop control
5020 36FF        LD      (HL),0FFH       Signal end of GAP's
5022 23          INC     HL              Bump to next entry in DCB
5023 10FB        DJNZ    5020H           Loop till DCB filled
5025 AF          XOR     A               Signal good status
5026 C9          RET                     Return to caller
*
*    -- Copy caller's file name to internal DCB. Validate name --
*
5027 060B        LD      B,0BH           Number of chars. to copy
5029 115D51      LD      DE,515DH        Internal DCB address
502C 3E20        LD      A,20H           ASCII blank
502E 12          LD      (DE),A          Move blank to internal DCB
502F 13          INC     DE              Bump to next internal DCB addr.
5030 10FC        DJNZ    502EH           Loop till internal DCB blank filled
5032 0608        LD      B,08H           Max. number of chars in file name
5034 115551      LD      DE,5155H        Address of password buffer
5037 12          LD      (DE),A          Blank fill password buffer
5038 13          INC     DE              One byte at a time
5039 10FC        DJNZ    5037H           Loop till buffer blank filled
503B 3EFF        LD      A,0FFH          Signal no drive spec (default)
503D 325451      LD      (5154H),A       (Reversed if drive specified)
5040 115D51      LD      DE,515DH        Addr. of Internal DCB
5043 0608        LD      B,08H           No. of chars. to move
5045 CD8150      CALL    5081H           Move file name to buffer at 515D
5048 4F          LD      C,A             C = last char.in name
5049 78          LD      A,B             A = 8 - number of chars. copied
504A FE08        CP      08H             If = 8, then no. chars. were copied
504C 2004        JR      NZ,5052H        Jump if file name was legitimate
504E 3E13        LD      A,13H           Error code for illegal char.
5050 B7          OR      A               Set status flags
5051 C9          RET                     And return to caller
5052 79          LD      A,C             Get terminating character
5053 FE2F        CP      2FH             Test for /
5055 2008        JR      NZ,505FH        Jump if no extension
5057 116551      LD      DE,5165H        Addr of ext. area in internal DCB
505A 0603        LD      B,03H           Max. number of chars. in extension
505C CD8150      CALL    5081H           Copy extension
505F FE2E        CP      2EH             Was terminating char. a .
                                         (password follows)
5061 2008        JR      NZ,506BH        Jump if not
5063 115551      LD      DE,5155H        Addr of internal password buffer
5066 0608        LD      B,08H           Maximum no. of chars. in password
5068 CD8150      CALL    5081H           Copy password to internal PW buffer
```

```
506B FE3A       CP      3AH             Was terminating char. a : (drive
                                        specification follows)
506D 2010       JR      NZ,507FH        Jump if no
506F 7E         LD      A,(HL)          Get drive number
5070 D630       SUB     30H             ASCII to binary
5072 3804       JR      C,5078H         Jump if drive number not a digit
5074 FE04       CP      04H             Test for drive no. greater than 3
5076 3804       JR      C,507CH         Jump if drive no. in range
(0-3)
5078 3E20       LD      A,20H           Error code for illegal drive no.
507A B7         OR      A               Set status flags for error.
507B C9         RET                     Return to caller
507C 325451     LD      (5154H),A       Save drive specification
507F AF         XOR     A               Signal good status (no error)
5080 C9         RET                     Return to caller
*
*       Copy characters pointed to by HL to location specified by
*       DE. Only 0-9 and A-Z will be copied. Terminate when a
*       non-eligible char. is found, or number of characters
*       specified in B have been copied.
*
5081 7E         LD      A,(HL)          Load a char. from  input string
5082 23         INC     HL              Bump to next char.
5083 1809       JR      508EH           Go qualify character
5085 7E         LD      A,(HL)          .   Load next character
5086 23         INC     HL              .   Bump to following character
5087 FE30       CP      30H             .   Is character at least a
                                        .   numeric
5089 D8         RET     C               .   Return if no
508A FE3A       CP      3AH             .   Character between 0-9
508C 3806       JR      C,5094H         .   Jump if yes
508E FE41       CP      41H             .   Not 0-9. Test for A-Z
5090 D8         RET     C               .   Return if between 9 and A
5091 FE5B       CP      5BH             .   Test if character above Z
5093 D0         RET     NC              .   Return if above Z. Not
                                        .   alphabetic
5094 12         LD      (DE),A          .   Move char.to our DCB
5095 13         INC     DE              .   Bump to next addr. in DCB
5096 10ED       DJNZ    5085H           .   Loop till all characters
                                        .   copied
5098 7E         LD      A,(HL)          Fetch next char.from input
                                        string
5099 23         INC     HL              Bump to next char.  in input
                                        string
509A C9         RET                     Return to caller
*
*     Compute hash code. Exclusive OR each character of file name
*     with the following character. Multiply result by 2 and use
*     as current character.
*
509B 060B       LD      B,0BH           Iteration count
509D 0E00       LD      C,00H           Initialize a mark value
509F 7E         LD      A,(HL)          .   Get a char from file name
50A0 23         INC     HL              .   Bump to next char. in name
50A1 A9         XOR     C               .   Exclusive OR with floating mask
50A2 07         RLCA                    .   Result times 2
50A3 4F         LD      C,A             .   becomes new floating mask
50A4 10F9       DJNZ    509FH           .   Loop till 11 characters of
                                        .   file name masked
50A6 79         LD      A,C             A = Final mask value
50A7 B7         OR      A               Set status flags
50A8 C0         RET     NZ              OK if non-zero
50A9 3C         INC     A               Else force a value of 1
50AA C9         RET                     and exit
*
*             Find an available hole in HIT sector.
*
50AB 3A4040     LD      A,(4040H)       Choose a random number
50AE E6E7       AND     0E7H            Force it to 00-07, 20-27, 40-47,
50B0 FE40       CP      40H             Then test if below 40
50B2 3002       JR      NC,50B6H        Jump if not, else
50B4 F680       OR      80H             force it 80-87, A0-A7, C0-C7, or E0-E7
50B6 6F         LD      L,A             Form beginning search address in HL
50B7 CDBD50     CALL    50BD            Search HIT, look for available slot
50BA C8         RET     Z               Exit if one found
50BB 2E40       LD      L,40H           Search again from start of table
50BD 7E         LD      A,(HL)          .   Get a HIT entry
50BE B7         OR      A               .   Set status floor
50BF C8         RET     Z               .   Exit if avail.byte found
50C0 7D         LD      A,L             .   Bump to first
50C1 C620       ADD     A,20H           .   Byte of next set of entries
50C3 6F         LD      L,A             .   Reform address
50C4 30F7       JR      NC,50BDH        .   Jump if end of table not reached
50C6 C641       ADD     A,41H           Restart search at 2, 3, ... of a set.
50C8 6F         LD      L,A             Reform addr in range 4X-47, 6X-67,...
50C9 E6E7       AND     0E7H            where X=1,2,...7
50CB BD         CP      L               If addr LSB = E7, found end of table
50CC 28EF       JR      Z,50BDH         Jump if not end of table
50CE F601       OR      01H             Signal no available entries
50D0 C9         RET                     Return to caller
*
*       Compute encode of password in buffer at 5155.
*       On entry DE = addr. of password buffer
*
50D1 21FFFF     LD      HL,0FFFFH       Mask value used for encode
50D4 0608       LD      B,08H           Max.no.chars. to encode
50D6 7B         LD      A,E             A = LSB of password buffer
50D7 C607       ADD     A,07H           Add 7 to position to end
50D9 5F         LD      E,A             of buffer. Must test for CARRY
50DA 3001       JR      NC,50DDH        And bump MSB if true
50DC 14         INC     D               Bump MSB of password buffer
50DD 1A         LD      A,(DE)          Fetch char. from password buffer
50DE D5         PUSH    DE              Save current buffer address
50DF 57         LD      D,A             Save char. to be encoded
50E0 5C         LD      E,H             E = Mark Value 1 (MV1)
50E1 7D         LD      A,L             A = Mark Value 2 (MV2)
50E2 E607       AND     07H             Reform MV2 by Exclusive ORing
50E4 0F         RRCA                    lower 3 bits and upper 3 bits
50E5 0F         RRCA                    Reposition lower 3 bits
50E6 0F         RRCA                    to upper 3 bits
50E7 AD         XOR     L               Exclusive OR lower and upper 3 bits
50E8 6F         LD      L,A             And save as new MV2
50E9 2600       LD      H,00H           Clear H to multiply new MV2 by 16
50EB 29         ADD     HL,HL           MV2 * 2
50EC 29         ADD     HL,HL           MV2 * 4
50ED 29         ADD     HL,HL           MV2 * 8
50EE 29         ADD     HL,HL           MV2 * 16
50EF AC         XOR     H               Combine upper 8 bits of MV2 * 16
50F0 AA         XOR     D               and partially reformed MV2 and
                                        current character
50F1 57         LD      D,A             Save upper 8 bits of encode
50F2 7D         LD      A,L             Preserve lower 8 bits of MV2 * 16
50F3 29         ADD     HL,HL           Compute MV2 * 32
50F4 AC         XOR     H               Combine upper 8 bits of MV2 * 32
50F5 AB         XOR     E               With lower 8 bits of MV2 * 16 and MV1
50F6 5F         LD      E,A             Save lower 8 bits of encode
50F7 EB         EX      DE,HL           HL = current encode
50F8 D1         POP     DE              Password buffer address
50F9 1B         DEC     DE              Skip down to next char.
50FA 10E1       DJNZ    50DDH           Loop till all characters encoded
50FC C9         RET                     Return to caller
*
*    Determine drive availability. Select drive and wait for
*    index. If no index detected within approx. 1 sec. there is
*    no drive, or the drive is not ready.
*
50FD 3ED0       LD      A,0D0H          Clear controller by sending it
50FF 32EC37     LD      (37ECH),A       a force interrupt command
5102 CD0046     CALL    4600H           Select unit in C-reg.
5105 C5         PUSH    BC              Save caller's BC
5106 015531     LD      BC,3155H        Count for approx.1 sec.
5109 CD1E51     CALL    511EH           Get disk status
510C 20FB       JR      NZ,5109H        Loop till index
510E CD1E51     CALL    511EH           Now
5111 28FB       JR      Z,510EH         wait 1 revolution
5113 CD1E51     CALL    511EH           And test for index again
5116 20FB       JR      NZ,5113H        Loop till 2nd index
5118 C1         POP     BC              Restore caller's BC
5119 07         RLCA                    This code is meaningless
511A E680       AND     80H             and will always
511C 87         ADD     A,A             Return a zero in the A-register
511D C9         RET                     Return to caller
*
*       Decrement count in BC while waiting for INDEX. If count
*       goes to zero before INDEX detected, return to caller of
*       50FD with a non-zero status.
*
511E 0B         DEC     BC              Decrement count
511F 78         LD      A,B             Combine both halves
5120 B1         OR      C               Set status flag
5121 2806       JR      Z,5129H         .   Go if count reached zero (error)
5123 3AEC37     LD      A,(37ECH)       .   Else get click status
5126 CB4F       BIT     01H,A           .   Test for index
5128 C9         RET                     .   And return to caller
5129 C1         POP     BC              .   Clear stack
512A C1         POP     BC              Clear stack
512B F601       OR      01H             Return to
512D C9         RET                     50FD caller w/error status
*
*     --- Read HIT sector for drive sqecified in C register ---
*
512E C5         PUSH    BC              Save caller's
512F D5         PUSH    DE              BC and DE
5130 CD554B     CALL    4B55H           Get directory into D reg.
5133 1E01       LD      E,01H           E = sector number for HIT
5135 21004D     LD      HL,4D00H        Buffer address
5138 CD354B     CALL    4B35H           Read HIT sector from directory track
513C D1         POP     DE              Restore caller's
513C C1         POP     BC              BC and DE
513D C8         RET     Z               If no error return with good status
513E 3E16       LD      A,16H           Else signal HIT read error
5140 C9         RET                     Then return to caller
*
*     --- Write HIT sector for drive specified in C register ---
*
5141 C5         PUSH    BC              Save caller's
5142 D5         PUSH    DE              BC and DE registers
5143 CD554B     CALL    4B55H           Get directory track number
                                        for unit in C-reg.
5146 1E01       LD      E,01H           Specify sector 1 (HIT sector)
5148 21004D     LD      HL,4D00H        Address of HIT sector buffer
514B CDEF46     CALL    46EFH           Write HIT sector
514E D1         POP     DE              Restore caller's
514F C1         POP     BC              BC and DE
5150 C8         RET     Z               Exit if no error during write
5151 3E17       LD      A,17H           Else signal HIT write error
5153 C9         RET                     And return to caller
*
*
*
5154 00         DEFB    00B             Drive specification (FF if none)
*
*
*
5180 13         INC     DE              Bump to MSB of EOF.
5181 1A         LD      A,(DE)          Fetch MSB of EOF.
5182 77         LD      (HL),A          Copy to DCB
5183 DD7E08     LD      A,(IX+08H)      Fetch EOF address from DCB
5186 B7         OR      A               Set status flags
5187 C8         RET     Z               Exit if INIT or OPEN call
5188 DD7E0C     LD      A,(IX+0CH)
5188 DD350C     DEC     (IX+0CH)
518E B7         OR      A
518F C0         RET     NZ
5190 DD350D     DEC     (IX+0DH)
5193 C9         RET
```

```
*********************************************************
*                                                       *
*                       SYS3/SYS                        *
*                                                       *
*                 CLOSE, KILL Processing                *
*                                                       *
*********************************************************


4E00 DDE5       PUSH    IX          Save DCB address
4E02 E5         PUSH    HL          Save buffer address
4E03 F5         PUSH    AF          Save request code
*
* Validate address of DCB and buffer. Validate drive
* number, HIT index and beginning track number
*
4E04 7A         LD      A,D         A = MSB of DCB addr.
4E05 FE40       CP      40H         Is DCB in system area?
4E07 385C       JR      C,4E65H     Error if yes
4E09 214A40     LD      HL,404AH    Addr. of MSB of mem. size
4E0C BE         CP      (HL)        Is DCB at end of memory?
4E0D 2802       JR      Z,4E11H     Error if yes
4E0F 3054       JR      NC,4E65H    Error if DCB addr. above mem. size
4E11 1A         LD      A,(DE)      Load 1st byte of DCB
4E12 CB7F       BIT     07H,A       Is file open
4E14 284F       JR      Z,4E65H     Error if not
4E16 D5         PUSH    DE          DCB addr. to stack
4E17 DDE1       POP     IX          Then load it into IX
4E19 DD7E04     LD      A,(IX+04H)  A = MSB of buffer addr.
4E1C FE40       CP      40H         Is buffer in system area
4E1E 3845       JR      C,4E65H     Error if yes
4E20 BE         CP      (HL)        Is buffer start at end of memory
4E21 2802       JR      Z,4E25H     Error if so, or if buffer
4E23 3040       JR      NC,4E65H    start above end of memory
4E25 DD7E06     LD      A,(IX+06H)  Load drive number
4E28 FE00       CP      00H         Test for illegal drive number
4E2A 3839       JR      C,4E65H     Error if drive out of range
4E2C FE04       CP      04H         Compare with max. drive number
4E2E 3035       JR      NC,4E65H    Error if drive out of range
4E30 DD7E07     LD      A,(IX+07H)  Load HIT index
4E33 CB67       BIT     04H,A       Test unused bit (should be 0)
4E35 202E       JR      NZ,4E65H    Error if on (HIT) = 10)
4E37 CB5F       BIT     03H,A       Test unused bit (should be 0)
4E39 202A       JR      NZ,4E65H    Error if on (HIT) = 08)
4E3B DD7E0E     LD      A,(IX+0EH)  Load track number 1st GAP
4E3E FE00       CP      00H         Are any tracks assigned
4E40 200A       JR      NZ,4E4CH    Jump if yes
4E42 DD7E0F     LD      A,(IX+0FH)  No, fetch gran count (may be track 0)
4E45 FE20       CP      20H         If track 0, must be 2nd granule
4E47 381C       JR      C,4E65H     Error if 1st granule (track 0,
                                    sector 0 reserved for system use)
4E49 DD7E0E     LD      A,(IX+0EH)  Reload track number 1st GAP
4E4C FEFF       CP      0FFH        Test for no tracks assigned
4E4E 2804       JR      Z,4E54H     Jump if none assigned
4E50 FE23       CP      23H         Greater than max. no. of tracks
4E52 3011       JR      NC,4E65H    Error if track greater than 34
*
*     Parameter validation complete. Test for
*     CLOSE or KILL option.
*
4E54 F1         POP     AF          Restore request code
4E55 E1         POP     HL          Restore buffer address
4E56 DDE1       POP     IX          Restore DCB address
4E58 E670       AND     70H         Isolate option code
4E5A FE10       CP      10H         Test for CLOSE request
4E5C CA6D4E     JP      Z,4E6DH     Jump if CLOSE call
4E5F FE20       CP      20H         Test for KILL request
4E61 CAD04F     JP      Z,4FD0H     Jump if KILL call
4E64 C9         RET                 Neither, ignore request
4E65 F1         POP     AF          Various errors. Restore req. code
4E66 E1         POP     HL          Buffer addr.
4E67 DDE1       POP     IX          And DCB addr.
4E69 3E26       LD      A,26H       Error code for file not opened
4E6B B7         OR      A           Set status flags
4E6C C9         RET                 Return with error status
*
*              ---- CLOSE Processing ----
*
4E6D CD9248     CALL    4892H       Save caller's registers
4E70 C0         RET     NZ          Error exit if file not open
4E71 CD7848     CALL    4878H       Flush sector buffer if necessary
4E74 C0         RET                 Exit if error while purging buffer
4E75 DD4607     LD      B,(IX+07H)  B = HIT Index (Dir sector no.)
4E78 DD4E06     LD      C,(IX+06H)  C = Unit
4E7B CDC14A     CALL    4AC1H       Read directory entry into 4D00.
                                    HL = addr. of entry
4E7E C0         RET     NZ          Exit if error during read
4E7F 3E03       LD      A,03H       Offset to EOF byte
4E81 85         ADD     A,L         Add offset to addr. of entry
4E82 6F         LD      L,A         Reform addr.
4E83 E5         PUSH    HL          Save addr. of EOF byte
4E84 DD7E08     LD      A,(IX+08H)  A = EOF offset from DCB
4E87 BE         CP      (HL)        DCB EOF = Dir entry EOF?
4E88 2014       JR      NZ,4E9EH    No, go update dir entry EOF
4E8A 3E11       LD      A,11H       Inc to EOF sector no. in directory
4E8C 85         ADD     A,L         Add to current directory address
4E8D 6F         LD      L,A         Reform directory address
4E8E DD7E0C     LD      A,(IX+0CH)  LSB of EOF sector from DCB
4E91 BE         CP      (HL)        Compare to LSB of EOF sector
                                    from directory
4E92 200A       JR      NZ,4E9EH    If unequal, update dir EOF sector #
4E94 2C         INC     L           Bump to MSB of EOF sector # in dir
4E95 DD7E0D     LD      A,(IX+0DH)  MSB of EOF sector from DCB
4E98 BE         CP      (HL)        Compare to MSB of EOF sector in dir
4E99 2003       JR      NZ,4E9EH    If unequal, update dir EOF sector #
4E9B F1         POP     AF          Clear stack
4E9C 1818       JR      4EB6H       Go compute total no. of grans
4E9E E1         POP     HL          Restore base address of
                                    directory entry + 3
4E9F DD7E08     LD      A,(IX+08H)  Load EOF byte offset from DCB
4EA2 77         LD      (HL),A      And copy to directory
4EA3 3E11       LD      A,11H       Offset to EOF sector no. in dir
4EA5 85         ADD     A,L         Add to current directory address
4EA6 6F         LD      L,A         And reform address in HL
4EA7 DD7E0C     LD      A,(IX+0CH)  Fetch LSB of ending sector
                                    number from DCB
4EAA 77         LD      (HL),A      Move it to directory entry
4EAB 2C         INC     L           Bump to addr. of MSB of ending
                                    sector in directory
4EAC DD7E0D     LD      A,(IX+0DH)  Fetch MSB of ending sector no.
4EAF 77         LD      (HL),A      And copy to directory
4EB0 E5         PUSH    HL          Save current directory address
4EB1 CDD64A     CALL    4AD6H       Write  updated directory entry
4EB4 E1         POP     HL          Restore dir. addr.
4EB5 C0         RET     NZ          Exit if error during write
4EB6 2C         INC     L           Bump to track number for 1st GAP
4EB7 110000     LD      DE,0000H    Zero storage for total granules
4EBA 7E         LD      A,(HL)      Fetch starting track no. from a GAP
4EBB 2C         INC     L           Bump pointer to granule count
4EBC FEFE       CP      0FEH        Test for overflow or end of GAPS
4EBE 300C       JR      NC,4ECCH    Jump if overflow or end of GAPS
4EC0 7E         LD      A,(HL)      A=No. of granules
4EC1 2C         INC     L           Bump to 1st byte of next GAP
4EC2 E61F       AND     1FH         Isolate count of granules
4EC4 3C         INC     A           Gives true value
4EC5 83         ADD     A,E         Accumulate total no. of grans in DE
4EC6 5F         LD      E,A         LSB of granule total
4EC7 30F1       JR      NC,4EBAH    Jump if LSB not over 255 yet
4EC9 14         INC     D           If it is, bump MSB
4ECA 18EE       JR      4EBAH       Loop till all granules summed
4ECC 200B       JR      NZ,4ED9H    Jump if end of GAPS reached
4ECE 46         LD      B,(HL)      Load HIT index for dir overflow
4ECF CDC14A     CALL    4AC1H       Read overflow directory entry
4ED2 C0         RET     NZ          Exit if error during read
4ED3 7D         LD      A,L         Get LSB of directory address
4ED4 C616       ADD     A,16H       Then add 16 to reach GAP address
4ED6 6F         LD      L,A         Reform addr. of GAPS in HL
4ED7 18E1       JR      4EBAH       Scan till end of GAPS reached
*
*
*     Test for change in file size using following method:
*     Directory file size (granule total from directory
*     GAP's) (minus) true file size (granule total from
*     ending sector in *DCB/5)
*
*     yields:
*
*     (Minus, if file has increased)
*     (zero, if file size is unchanged)
*     (positive, if file has decreased)
*
*
*
4ED9 E5         PUSH    HL          Save current addr in dir. End of GAPS
4EDA DD6E0C     LD      L,(IX+0CH)  L = LSB of number of records
4EDD DD660D     LD      H,(IX+0DH)  H = MSB of number of records
4EE0 3E05       LD      A,05H       A = Divisor
4EE2 CD844B     CALL    4B84H       Modulo divide HL by 5
4EE5 23         INC     HL          Quotient + 1
                                    (no. of grans = true size)
4EE6 AF         XOR     A           Clear carry
4EE7 EB         EX      DE,HL       HL = # of granules in directory.
                                    DE = true file size
4EE8 ED52       SBC     HL,DE       Compare true file size (by record
                                    count in DCB) to directory file
4EEA EB         EX      DE,HL       size (by GAP's). Save diff. in DE
4EEB E1         POP     HL          Restore current addr. in
                                    directory. End of GAPS
4EEC CA7B4F     JP      Z,4F7BH     If dir gran count is equal to or
4EEF DA7B4F     JP      C,4F7BH     greater, don't update directory
4EF2 2D         DEC     L           Backspace to granule count
4EF3 2D         DEC     L           For last GAP
4EF4 E5         PUSH    HL          Save addr of last good GAP in dir
4EF5 D5         PUSH    DE          Save granule difference
4EF6 CDF04A     CALL    4AF0H       Read GAT sector into 4D00
4EF9 2003       JR      NZ,4EFEH    Jump if error during read. Skip
                                    read into 5100
4EFB CD6850     CALL    5068H       Read HIT sector into 5100
4EFE D1         POP     DE          Restore granule difference
4EFF E1         POP     HL          Restore addr of last GAP in dir
4F00 C0         RET     NZ          Exit if GAT read error either time
4F01 7B         LD      A,E         Load LSB of granule difference
4F02 FE00       CP      00H         Test if done
4F04 2869       JR      Z,4F6FH     Jump if done
4F06 D5         PUSH    DE          Save current granule difference
4F07 7E         LD      A,(HL)      Fetch granule count from last
                                    GAP in directory
4F08 E6E0       AND     0E0H        Isolate starting gran no. (bit 5)
4F0A 07         RLCA                Shift into low order of A register
4F0B 07         RLCA                Gives 0 if file starts on sector 0
4F0C 07         RLCA                of first track, or 1 if file begins
4F0D 5F         LD      E,A         on sector 5 of first track
4F0E 7E         LD      A,(HL)      Refetch gran count from last GAP
4F0F E61F       AND     1FH         Isolate no. of assigned grans
4F11 83         ADD     A,E         Add offset for 1st gran, save in E
4F12 5F         LD      E,A         Adds 0 if starting sect # is 0
                                    Adds 1 if starting sect # is 5
4F13 E6FE       AND     0FEH        Clear bit 0, so division will
                                    not affect carry
4F15 0F         RRCA                Divide granule count by 2
4F16 32224F     LD      (4F22H),A   And save number of tracks
4F19 7B         LD      A,E         Restore original gran count & offset
4F1A E601       AND     01H         Isolate Bit 0
```

```
*
*  Construct mask for clearing bit in GAT for last gran assigned
*  Use following rules:
*
*        ---------------------------------------
*        | # of sectors in    sector      offset |
*        | granule count        0           1    |
*        ---------------------------------------
*        | EVEN                FE          FD    |
*        ---------------------------------------
*        | ODD                 FD          FE    |
*        ---------------------------------------
*
4F1C 3C         INC     A               Then add 1 and compliment
4F1D 2F         CPL                     to generate a mark for clearing
4F1E F5         PUSH    AF              the last assigned granule in
                                        the GAT. Save mask
4F1F 2B         DEC     HL              Backspace GAP pointer to track no.
4F20 7E         LD      A,(HL)          Fetch starting track number for GAP
4F21 C600       ADD     A,00H           Add no. of consecutively assigned
                                        tracks. Gives ending track #
4F23 EB         EX      DE,HL           Save directory GAP addr. in DE
4F24 264D       LD      H,4DH           Form address of GAT entry for
4F26 6F         LD      L,A             ending track in HL
4F27 F1         POP     AF              Restore mask used to clear last
                                        granule assigned
4F28 A6         AND     (HL)            Clear bit for last gran assigned
4F29 77         LD      (HL),A          And restore byte with cleared
                                        bit to GAT buffer
*
*    Look for GAPS which have no granules assigned. When one
*    is found, set both bytes of the GAP entry to FF. If all
*    GAP's in an entry are released (set to FF's), release the
*    directory entry and the HIT index in the HIT sector for
*    the directory entry.
*
4F2A EB         EX      DE,HL           Restore directory GAP addr to HL
4F2B 23         INC     HL              Skip forward to number of
                                        granules assigned this GAP
4F2C 35         DEC     (HL)            Prepare to test for no grans assigned
4F2D 7E         LD      A,(HL)          Fetch # granules assigned - 2
4F2E 3C         INC     A               Add 1. Yields  zero if none assigned
4F2F E61F       AND     1FH             Isolate true gran count for GAP
4F31 D1         POP     DE              Restore difference in granule
                                        count between DCB & dir.
4F32 2038       JR      NZ,4F6CH        Jump if at least one gran assigned
4F34 36FF       LD      (HL),0FFH       No granules assigned to this GAP
4F36 2B         DEC     HL              Initialize GAP to FF
4F37 36FF       LD      (HL),0FFH       Including track number
4F39 2B         DEC     HL              Backspace to gran count for last GAP
4F3A 7D         LD      A,L             Then isolate GAP index
4F3B E61F       AND     1FH             And test for start of GAP
4F3D FE15       CP      15H             entries in directory. If not
4F3F 202B       JR      NZ,4F6CH        start of GAP area - Go
4F41 AD         XOR     L               Form LSB of dir entry addr in A
4F42 6F         LD      L,A             Then reform start of directory
                                        entry address in HL
4F43 7E         LD      A,(HL)          Fetch access and dir type flags
4F44 3600       LD      (HL),00H        Flag directory entry as available
4F46 23         INC     HL              Bump pointer to overflow byte
                                        pointer in directory
4F47 56         LD      D,(HL)          Fetch possible overflow pointer
4F48 2651       LD      H,51H           Origin of HIT buffer to H
4F4A 68         LD      L,B             HIT index for file to HL.
                                        HL = addr.of HIT index
4F4B 3600       LD      (HL),00H        Clear HIT entry for dir entry
4F4D CB7F       BIT     07H,A           Test type of dir entry being cleared
4F4F 281E       JR      Z,4F6FH         Jump if primary directory entry
4F51 D5         PUSH    DE              Save remainder of gran count diff
4F52 CDD64A     CALL    4AD6H           Write released directory entry
4F55 D1         POP     DE              Restore granule count difference
4F56 C0         RET     NZ              Exit if error during sector rewrite
4F57 78         LD      A,B             Fetch HIT index and compare
4F58 AA         XOR     D               Sector no. portion of it with
4F59 E607       AND     07H             the sector no. from overflow pointer
4F5B 42         LD      B,D             Save sector number for next
                                        overflow pointer
4F5C C4C14A     CALL    NZ,4AC1H        Read overflow directory entry
                                        (if there is one)
4F5F C0         RET     NZ              Exit if error during read
4F60 78         LD      A,B             A = entry addr/sector no. -2
4F61 E6E0       AND     0E0H            Isolate directory entry address
4F63 C61F       ADD     A,1FH           Skip to end of GAP's in overflow
                                        directory entry
4F65 6F         LD      L,A             Form ending dir addr in HL
4F66 36FF       LD      (HL),0FFH       Signal end of GAP's
4F68 2B         DEC     HL              and backspace to starting track no.
4F69 36FF       LD      (HL),0FFH       Initialize it on end of GAP's
4F6B 2B         DEC     HL              Backspace to gran count for next GAP
4F6C 1D         DEC     E               Reduce difference in gran count
4F6D 1892       JR      4F01H           and go test if done
*
*     Write updated directory entry. Reconstruct DCB
*
4F6F CDD64A     CALL    4AD6H           Write updated directory entry
4F72 C0         RET     NZ              Exit if error during write
4F73 CD7750     CALL    5077H           Maybe write updated HIT sector
4F76 C0         RET     NZ              Exit if error during HIT rewrite
4F77 CD034B     CALL    4B03H           Write GAT sector
4F7A C0         RET     NZ              Exit if error during write
4F7B DD7E07     LD      A,(IX+07H)      Load HIT index
4F7E A8         XOR     B               Compare with last HIT index
4F7F E61F       AND     1FH             Isolate any possible difference
4F81 C4C14A     CALL    NZ,4AC1H        They were different, read
                                        directory entry
4F84 C0         RET     NZ              Exit if error during read
4F85 DD7E06     LD      A,(IX+06H)      Load drive number
4F88 E603       AND     03H             Force a value between 0 and 3
4F8A F630       OR      30H             Reconstruct as ASCII value
4F8C 32C84F     LD      (4FC8H),A       to be restored into DCB
4F8F 2642       LD      H,42H           Form address of file name in dir
4F91 DD7E07     LD      A,(IX+07H)      Get the HIT index
4F94 E6E0       AND     0E0H            Isolate offset to file entry
4F96 F605       OR      05H             Add a 5 and move to L
4F98 6F         LD      L,A             Gives addr. of file name
                                        in directory entry
4F99 E5         PUSH    HL              Save addr. of file name in dir
4F9A DDE5       PUSH    IX              Start of DCB address to DE
4F9C D1         POP     DE              Prepare to rebuild DCB for OPEN call
4F9D 0608       LD      B,08H           Max. no. of chars in file name
4F9F 7E         LD      A,(HL)          Get a char. from file name
4FA0 FE20       CP      20H             Is it a blank
4FA2 2805       JR      Z,4FA9H         Jump if yes. File name copied
4FA4 12         LD      (DE),A          Move  char from directory to DCB
4FA5 23         INC     HL              Bump directory addr.
4FA6 13         INC     DE              Bump DCB addr.
4FA7 10F6       DJNZ    4F9FH           Loop till name copied to DCB
4FA9 E1         POP     HL              Restore addr of name in directory
4FAA 7D         LD      A,L             Then add 8 to it to
4FAB C608       ADD     A,08H           Get addr of extension in directory
4FAD 6F         LD      L,A             HL = Addr. of file ext.
4FAE 7E         LD      A,(HL)          Fetch a character of extension
4FAF FE20       CP      20H             Is it a blank
4FB1 2810       JR      Z,4FC3H         Jump if yes, no extension
4FB3 3E2F       LD      A,2FH           Else, save a "/"
4FB5 12         LD      (DE),A          After file name in DCB
4FB6 13         INC     DE              Bump to pos. of 1st ext. char in DCB
4FB7 0603       LD      B,03H           Max. no. of chars in extension
4FB9 7E         LD      A,(HL)          Fetch a character of extension
4FBA FE20       CP      20H             Is it a blank
4FBC 2805       JR      Z,4FC3H         Go if so. End of ext. found
4FBE 12         LD      (DE),A          Else save extension character
4FBF 23         INC     HL              Bump directory address
4FC0 13         INC     DE              Bump DCB address
4FC1 10F6       DJNZ    4FB9H           Loop till ext. copied to DCB
4FC3 3E3A       LD      A,3AH           Next, save an ASCII :
4FC5 12         LD      (DE),A          in DCB, then
4FC6 13         INC     DE              Bump to next addr.
4FC7 3E00       LD      A,00H           Load ASCII drive number
4FC9 12         LD      (DE),A          And save it in DCB
4FCA 13         INC     DE              Bump to first char after drive no.
4FCB 3E03       LD      A,03H           And store a
4FCD 12         LD      (DE),A          terminator
4FCE AF         XOR     A               Signal good status
4FCF C9         RET                     Return to caller
*
*                  ---- KILL Processing ----
*
4FD0 CD9248     CALL    4892H           Save caller's registers
4FD3 C0         RET     NZ              Error exit if file not open.
4FD4 DD7E01     LD      A,(IX+01H)      Get permission flags
4FD7 E607       AND     07H             Isolate them
4FD9 FE02       CP      02H             Test if KILL permission
4FDB 3804       JR      C,4FE1H         Jump if KILL permission. Else
4FDD 3E25       LD      A,25H           ret illegal file access code
4FDF B7         OR      A               Set error status flags
4FE0 C9         RET                     And return to caller
4FE1 DD4E06     LD      C,(IX+06H)      C = Unit no.
4FE4 DD4607     LD      B,(IX+07H)      B = HIT index number
4FE7 CD6850     CALL    5068H           Read HIT sector into 5100
4FEA C0         RET     NZ              Exit if error during read
4FEB CDF04A     CALL    4AF0H           Read GAT into 4D00
4FEE C0         RET     NZ              Exit if error
4FEF CDC14A     CALL    4AC1H           Read directory entry into 4200.
                                        HL = addr. of entry
4FF2 C0         RET     NZ              Exit if error during read
4FF3 3E16       LD      A,16H           Offset to start of GAP's
4FF5 85         ADD     A,L             Combine with directory entry
                                        address
4FF6 6F         LD      L,A             Form GAP address for dir. entry
                                        in HL
4FF7 5E         LD      E,(HL)          Fetch track address from GAP
4FF8 2C         INC     L               Bump pointer to granule count
4FF9 56         LD      D,(HL)          Fetch number of assigned
                                        granules
4FFA 7B         LD      A,E             Prepare to test for end of GAP's
4FFB FEFE       CP      0FEH            Test if end of GAP's, or
                                        overflow pointer reached
4FFD 3006       JR      NC,5005H        Jump if yes
4FFF 2C         INC     L               Else bump to next GAP entry
5000 CD3950     CALL    5039H           Deallocate GAP
5003 18F2       JR      4FF7H           Loop till end of GAP's reached,
                                        or overflow pointer found
5005 7D         LD      A,L             Get LSB of directory entry addr.
                                        in sector buffer
5006 E6E0       AND     0E0H            Isolate starting address of
                                        entry
5008 6F         LD      L,A             And form beginning address in HL
5009 3600       LD      (HL),00H        Clear access flags, flag entry
                                        as available
500B C5         PUSH    BC              Save HIT index/unit number
500C D5         PUSH    DE              Save GAP flags (FF or FE)
500D E5         PUSH    HL              Start of directory entry address
500E D1         POP     DE              To DE
500F 13         INC     DE              Bump to 2nd byte of directory
                                        entry address
5010 011F00     LD      BC,001FH        Number of bytes to move
5013 EDB0       LDIR                    Zero directory entry in sector
                                        buffer
5015 D1         POP     DE              Restore GAP terminator flag
5016 C1         POP     BC              Restore HIT index/unit number
5017 CDD64A     CALL    4AD6H           Write drectory sector with
                                        zeroed entry
501A C0         RET     NZ              Exit if error during write
501B 2651       LD      H,51H           Load MSB of HIT buffer address,
                                        then load HIT index
501D 68         LD      L,B             To form address of byte
                                        containing hash code for
                                        file
```

```
501E 3600       LD      (HL),00H        Remove hash code from HIT sector
                                        buffer
5020 42         LD      B,D             B = sector # of possible
                                        overflow entry
5021 7B         LD      A,E             Fetch GAP terminating flag
5022 FEFE       CP      0FEH            Test for overfow entry
5024 28C9       JR      Z,4FEFH         Jump if overflow entry. Go read
                                        and process
5026 CD034B     CALL    4B03H           Else, write updated GAT sector
                                        from buffer at 4D00
5029 C0         RET     NZ              Exit if error during write
502A CD7750     CALL    5077H           Write updated HIT sector from
                                        buffer at 5200
502D C0         RET     NZ              Exit if error during write
502E DDE5       PUSH    IX              Copy DCB address
5030 E1         POP     HL              to HL
5031 0620       LD      B,20H           Number of bytes to clear in DCB
5033 AF         XOR     A               Load a zero into A-reg.
5034 77         LD      (HL),A          Clear a byte in DCB
5035 23         INC     HL              Bump to next address in DCB
5036 10FC       DJNZ    5034H           Loop till DCB cleared
5038 C9         RET                     Return to KILL caller
*
*                    GAP Deallocation
*
5039 E5         PUSH    HL              Save current directory address
                                        GAP deallocation
503A C5         PUSH    BC              Save HIT index/unit number
503B 6B         LD      L,E             Track number to L. Will become
                                        index into GAT
503C 264D       LD      H,4DH           Form address for starting track
                                        number in HL
503E 7A         LD      A,D             Get number of consecutively
                                        assigned granules
503F E61F       AND     1FH             Isolate granuule count -1
5041 4F         LD      C,A             Save in C
5042 0C         INC     C               And compute true granule count
5043 AA         XOR     D               Isolate starting granule number
                                        flag (0 = sect. 0,
                                        1 = sect.5)
5044 07         RLCA                    And reposition
5045 07         RLCA                    Into
5046 07         RLCA                    Lower bit of A-reg.
5047 F5         PUSH    AF              Save it as the initial bit
                                        counter for each GAT entry
5048 46         LD      B,(HL)          Fetch current GAT entry for
                                        current track number
5049 CD5B50     CALL    505BH           Release a granule (make it
                                        available) for current GAT entry
504C 70         LD      (HL),B          Restore updated GAT entry
504D F1         POP     AF              Get bit counter for current GAT
                                        entry
504E 3C         INC     A               Bump it and
504F FE02       CP      02H             Test if Max. # granules released
                                        for this entry
5051 2002       JR      NZ,5055H        Jump more granules to be
                                        released for this GAT entry
5053 AF         XOR     A               Else, reset bit counter to zero
5054 2C         INC     L               Bump to GAT entry for next track
5055 0D         DEC     C               Count 1 granule released
5056 20EF       JR      NZ,5047H        Jump if more granules to be
                                        released
5058 C1         POP     BC              Restore HIT index, unit number
5059 E1         POP     HL              Restore current directory GAP
                                        address
505A C9         RET                     Return to caller
*
*       Clear bit number specified by A, in byte in B
*
505B E607       AND     07H             Isolate bit number to be reset.
                                        Will be 0 or 1
505D 07         RLCA                    And
505E 07         RLCA                    Position for
505F 07         RLCA                    Placement into RES instr.
5060 F680       OR      80H             Combine RES op code with bit to
                                        be cleared
5062 326650     LD      (5066H),A       Save synthesized RES instr.
5065 CB80       RES     00H,B           And execute it. Clear bit
                                        associated with granule
5067 C9         RET                     Return to caller
*
*                     Read HIT Sector
*
5068 CD554B     CALL    4B55H           Get directory track number into D
506B 1E01       LD      E,01H           Specify sector # (HIT sector)
506D 210051     LD      HL,5100H        HL = Buffer address of 5100
5070 CD354B     CALL    4B35H           Read GAT sector into 5100
5073 C8         RET     Z               Return if no error
5074 3E16       LD      A,16H           Signal HIT read error
5076 C9         RET                     Return to caller
*
*                    Write HIT Sector
*
5077 CD554B     CALL    4B55H           Get directory track number into D
507A 1E01       LD      E,01H           Specify sector #3 (HIT sector)
507C 210051     LD      HL,5100H        HL = Buffer address of 5100
507F CDEF46     CALL    46EFH           Write HIT sector
5082 C8         RET     Z               Return if no error
5083 3E17       LD      A,17H           Else, signal HIT write error
5085 C9         RET                     and return to caller
```

```
*************************************************************
*                                                           *
*                        SYS4/SYS                           *
*                  Error Message Processor                  *
*                                                           *
*************************************************************


4E00 E3          EX      (SP),HL         Return addr to HL. HL to stack
4E01 226E4E      LD      (4E6EH),HL      Return addr to exit instruction.
4E04 E1          POP     HL              Restore HL (clear stack)
4E05 F1          POP     AF              Restore error code
4E06 E5          PUSH    HL              Preserve HL
4E07 D5          PUSH    DE              DE
4E08 C5          PUSH    BC              BC
4E09 F5          PUSH    AF              And error code
4E0A 47          LD      B,A             Save original error code
4E0B E63F        AND     3FH             Clear upper two bits
4E0D 4F          LD      C,A             C = cleared error code
4E0E CB70        BIT     06H,B           Test if ***ERRCOD=XX message flag set
4E10 0600        LD      B,00H           BC = error code (used as index
4E12 2016        JR      NZ,4E2AH        Jump if ***ERRCOD msg flag off
*
*                  Hex to ASCII Conversion
*
4E14 1E30        LD      E,30H           E = ASCII zero
4E16 D60A        SUB     0AH             Subtract 10 from hex value
4E18 3803        JR      C,4E1DH         Jump if done with conversion
4E1A 1C          INC     E               Bump ASCII value
4E1B 18F9        JR      4E16H           Loop till ASCII equivalent computed
4E1D C63A        ADD     A,3AH           Make MSB of hex ASCII value positive
4E1F 57          LD      D,A             for ASCII error code in DE
4E20 ED53424F    LD      (4F42H),DE      Then save in ***ERRCOD=XX message
4E24 21364F      LD      HL,4F36H        Address of ***ERRCOD message
4E27 CD6744      CALL    4467H           Display msg.
4E2A 21844F      LD      HL,4F84H        Addr of error code index list
4E2D 09          ADD     HL,BC           Add error code to get address of
                                         phrase indices for error
4E2E 6E          LD      L,(HL)          L = LSB of phrase index address
                                         for error code
4E2F 2651        LD      H,51H           H = MSB of phrase index address
4E31 7E          LD      A,(HL)          Load a word index
4E32 E63F        AND     3FH             Clear terminator/unused lists
4E34 87          ADD     A,A             Index * 2 (list is addr by word
4E35 E5          PUSH    HL              Save current word index address
4E36 4F          LD      C,A             C = index * 2
4E37 0600        LD      B,00H           BC = index * 2
4E39 21C24F      LD      HL,4FC2H        Origin -2 of word address list
4E3C 09          ADD     HL,BC           Add index * 2 for required phrase
4E3D 5E          LD      E,(HL)          E = LSB of addr for required phrase
4E3E 23          INC     HL              Bump to address of MSB
4E3F 56          LD      D,(HL)          D = MSB of addr for required phrase
4E40 23          INC     HL              Bump to LSB of addr for next phrase
4E41 7E          LD      A,(HL)          Fetch it
4E42 23          INC     HL              Bump to MSB of addr for next phrase
4E43 66          LD      H,(HL)          Fetch it
4E44 6F          LD      L,A             HL = text addr for following phrase
4E45 B7          OR      A               Clear carry
4E46 ED52        SBC     HL,DE           Addr of next phrase (required phrase)
4E48 45          LD      B,L             Gives no. of chars in required phrase
4E49 EB          EX      DE,HL           HL = text addr for required phrase
4E4A 7E          LD      A,(HL)          Fetch a character from phrase
4E4B 23          INC     HL              Bump to next char in phrase
4E4C CD3300      CALL    0033H           Display phrase character
4E4F 10F9        DJNZ    4E4AH           Loop till all of phrase displayed
4E51 3E20        LD      A,20H           Follow phrase
4E53 CD3300      CALL    0033H           with a space
4E56 E1          POP     HL              Restore phrase index address
4E57 7E          LD      A,(HL)          Refetch index
4E58 23          INC     HL              Bump to the next index in case
                                         this is not the end
4E59 07          RLCA                    Terminating bit to CARRY flag
4E5A 30D5        JR      NC,4E31H        Jump if not end of phrase
                                         list. Process next phrase
4E5C F1          POP     AF              Restore error code
4E5D F5          PUSH    AF              Save error code
4E5E CB77        BIT     06H,A           Test if file error code
4E60 CC734E      CALL    Z,4E73H         Display file name if file error
4E63 3E0D        LD      A,0DH           Else scroll one line
4E65 CD3300      CALL    0033H           by displaying a carriage return
4E68 F1          POP     AF              Restore error code in A,
4E69 C1          POP     BC              then restore BC
4E6A D1          POP     DE              DE
4E6B E1          POP     HL              and HL
4E6C B7          OR      A               Set status flag for return
                                         Test for System or caller return
4E6D FA0000      JP      M,0000H         Return to caller if requested
4E70 C33040      JP      4030H           Else return to system
*
*
*
4E73 E5          PUSH    HL              Save last phrase index address
4E74 21474F      LD      HL,4F47H        Address of ***
4E77 CD6744      CALL    4467H           Display closing ***
4E7A E1          POP     HL              Restore last phrase index addr
4E7B DDE5        PUSH    IX              Save current system DCB address
4E7D DD2A0A43    LD      IX,(430AH)      Load DCB addr for file with error
4E81 2B          DEC     HL              Position to terminating character
                                         in phrase index list
4E82 CB76        BIT     06H,(HL)        Test if file name in DCB
4E84 201A        JR      NZ,4EA0H        Jump if file name in DCB
4E86 DD4E06      LD      C,(IX+06H)      C = drive number
4E89 DD4607      LD      B,(IX+07H)      B = directory sector number
4E8C DDCB007E    BIT     07H,(IX+00H)    Test if file OPEN
4E90 2032        JR      NZ,4EC4H        If so, go read dir - copy name/ext
4E92 DDE1        POP     IX              Else, clear stack
4E94 ED43554F    LD      (4F55H),BC      Save device code in DEVICE = xx msg
4E98 214B4F      LD      HL,4F4BH        Address of DEVICE = xx msg
4E9B CD6744      CALL    4467H           Display message
4E9E 1866        JR      4F06H           Go display REFERENCED AT message
                                         and return to caller
4EA0 DD7E00      LD      A,(IX+00H)      Test for special DCB code
4EA3 FE2A        CP      2AH             First character must be *
4EA5 2008        JR      NZ,4EAFH        Jump if not. Name is in DCB
4EA7 DD4E01      LD      C,(IX+01H)      Special DCB
4EAA DD4602      LD      B,(IX+02H)      Bytes 2 - 3 contain device code
4EAD 18E3        JR      4E92H           Go print DEVICE=XX msg
4EAF DDE5        PUSH    IX              Save DCB address belonging to
                                         file with error
4EB1 E1          POP     HL              HL = DCB address with file name
4EB2 11604F      LD      DE,4F60H        Address of file name in
                                         <file = NNN...> msg
4EB5 011800      LD      BC,0018H        Max number of chars in name/ext:D
4EB8 7E          LD      A,(HL)          Fetch a char from file name
4EB9 FE03        CP      03H             Test for end of list
4EBB 283C        JR      Z,4EF9H         Jump if end of list
4EBD 23          INC     HL              Else bump to next char in list
4EBE 12          LD      (DE),A          Move last char to <file = NNN...> msg
4EBF 13          INC     DE              Bump to next position in file message
4EC0 10F6        DJNZ    4EB8H           Loop till file/ext:drive copied
                                         into <file = ...> message
4EC2 1835        JR      4EF9H           Go display <file = ...> message
4EC4 CDC14A      CALL    4AC1H           Read dir entry for file with error
4EC7 010500      LD      BC,0005H        Add offset to get address
4ECA 09          ADD     HL,BC           of file name in directory
4ECB 11604F      LD      DE,4F60H        Addr of file name in <file = ...> msg
4ECE 0608        LD      B,08H           Max. no. chars to copy from dir
4ED0 7E          LD      A,(HL)          Get file name char from directory
4ED1 FE20        CP      20H             Is it blank (end of name reached)
4ED3 2805        JR      Z,4EDAH         Jump if end of name
4ED5 2C          INC     L               Bump to next char if not
4ED6 12          LD      (DE),A          Save this char in <file = ...> msg
4ED7 13          INC     DE              Bump to next pos in <file = ...> msg
4ED8 10F6        DJNZ    4ED0H           Loop till 8 characters copied
                                         or end of name found
4EDA 3E2F        LD      A,2FH           Then load an ASCII "/"
4EDC 12          LD      (DE),A          Put after file name in <file = ...> msg
4EDD 13          INC     DE              Pos to addr of ext in <,file> msg
4EDE 48          LD      C,B             C = number of characters
                                         remaining in name field
4EDF 0600        LD      B,00H           So BC can be added to HL
4EE1 09          ADD     HL,BC           compute beginning addr of ext
4EE2 0603        LD      B,03H           Max no. chars in extension
*
*
*
4EE4 7E          LD      A,(HL)          .<-- Fetch a char from ext
4EE5 2C          INC     L               .   Bump to next char in ext
4EE6 FE20        CP      20H             .   If char blank, we're done
4EE8 2804        JR      Z,4EEEH         .   Jump if end of ext found
4EEA 12          LD      (DE),A          .   Move ext char to message
4EEB 13          INC     DE              .   Bump message store address
4EEC 10F6        DJNZ    4EE4H           .-- Loop till extension copied
4EEE EB          EX      DE,HL           HL = addr of drive no. in <file = ...>
4EEF 363A        LD      (HL),3AH        terminate filename/ext
4EF1 23          INC     HL              with a :
4EF2 DD7E06      LD      A,(IX+06H)      Then fetch drive from DCB
4EF5 C630        ADD     A,30H           and convert it to ASCII
4EF7 77          LD      (HL),A          Save in <file = ...> message
4EF8 23          INC     HL              Bump to pos after drive # in msg
4EF9 363E        LD      (HL),3EH        Save closing >
4EFB 23          INC     HL              Bump past > in msg
4EFC 360D        LD      (HL),0DH        Terminate with carriage return
4EFE DDE1        POP     IX              Restore current DCB address
4F00 21594F      LD      HL,4F59H        Address of <file = XXX/XX:d> msg
4F03 CD6744      CALL    4467H           Display message
4F06 21704F      LD      HL,4F70H        Address of REFERENCED AT message
4F09 CD6744      CALL    4467H           Display message
4F0C 2A0C43      LD      HL,(430CH)      HL = sector buffer address
                                         associated with file
4F0F 2B          DEC     HL              Decrement buffer
4F10 2B          DEC     HL              Address
4F11 2B          DEC     HL              By three
4F12 CD1B4F      CALL    4F1BH           Convert addr to ASCII and display
4F15 21824F      LD      HL,4F82H        Address of ' string (terminate
                                         buffer address display)
4F18 C36744      JP      4467H           Display ending ', ret to caller
4F1B 7C          LD      A,H             Convert MSB of buffer addr first
4F1C CD204F      CALL    4F20H           Call routine to convert and
                                         display value in A reg
4F1F 7D          LD      A,L             Now convert/display LSB of buff addr
4F20 F5          PUSH    AF              Save value to be converted
4F21 1F          RRA                     Right shift hex
4F22 1F          RRA                     value four times
4F23 1F          RRA                     This is the same
4F24 1F          RRA                     as dividing it by 16
4F25 CD294F      CALL    4F29H           Then convert binary hex value
                                         to ASCII and display
4F28 F1          POP     AF              Restore original value
4F29 E60F        AND     0FH             Isolate binary field to convert
4F2B C630        ADD     A,30H           Gives ASCII equivalent of binary
4F2D FE3A        CP      3AH             Test if value exceeds 9
4F2F 3802        JR      C,4F33H         Jump if not, value less than 10
4F31 C607        ADD     A,07H           Add 7 to get A-F ASCII value
4F33 C33300      JP      0033H           Display char. Return to caller
4F36             DEFB    0AH             Line feed
4F37             DEFM    '*** ERRCOD = XX , '
4F46             DEFB    03H             Message terminator
4F47             DEFM    '***'
4F4A             DEFB    0DH             Message terminator
4F4B             DEFM    '<DEVICE = *XX>
4F58             DEFB    0DH             Message terminator
4F59             DEFM    '<FILE =NNNNNNNN/EEE:d>
4F6F             DEFB    0DH             Message terminator
4F70             DEFM    'REFERENCED AT X'
4F80             DEFB    27H             Hex valve for
4F81             DEFB    03H             Message terminator
4F82             DEFB    27H             Hex valve for
4F83             DEFB    0DH             Message terminator
```

```
*
*               Error Number Message Addresses
*
4F84            DEFB    5BH             00, 515B
4F85            DEFB    5DH             01, 515d
4F86            DEFB    62H             02, 5162
4F87            DEFB    66H             03, 5166
4F88            DEFB    6AH             04, 516A
4F89            DEFB    6EH             05, 516E
4F8A            DEFB    74H             06, 5174
4F8B            DEFB    79H             07, 5179
4F8C            DEFB    7EH             08, 517E
4F8D            DEFB    81H             09, 5181
4F8E            DEFB    86H             10, 5186
4F8F            DEFB    8AH             11, 518A
4F90            DEFB    8EH             12, 518E
4F91            DEFB    92H             13, 5192
4F92            DEFB    98H             14, 5198
4F93            DEFB    9DH             15, 519D
4F94            DEFB    A0H             16, 51A0
4F95            DEFB    A4H             17, 51A4
4F96            DEFB    A7H             18, 51A7
4F97            DEFB    AAH             19, 51AA
4F98            DEFB    ADH             20, 51AD
4F99            DEFB    B0H             21, 51B0
4F9A            DEFB    B3H             22, 51B3
4F9B            DEFB    B6H             23, 51B6
4F9C            DEFB    B9H             24, 51B9
4F9D            DEFB    BDH             25, 51BD
4F9E            DEFB    C0H             26, 51C0
4F9F            DEFB    C3H             27, 51C3
4FA0            DEFB    C6H             28, 51C6
4FA1            DEFB    CAH             29, 51CA
4FA2            DEFB    CFH             30, 51CF
4FA3            DEFB    D3H             31, 51D3
4FA4            DEFB    D6H             32, 51D6
4FA5            DEFB    D9H             33, 51D9
4FA6            DEFB    DDH             34, 51DD
4FA7            DEFB    E1H             35, 51E1
4FA8            DEFB    E3H             36, 51E3
4FA9            DEFB    E8H             37, 51E8
4FAA            DEFB    EDH             38, 51ED
4FA1            DEFB    F0H             39, 51F0
4FA2            DEFB    F0H             30, 51F0
4FA3            DEFB    F0H             30, 51F0
4FA4            DEFB    F0H             30, 51F0
4FA5            DEFB    F0H             30, 51F0
4FA6            DEFB    F0H             30, 51F0
4FA7            DEFB    f0H             30, 51F0
4FA8            DEFB    F0H             30, 51F0
4FA9            DEFB    F0H             30, 51F0
4FaA            DEFB    F0H             30, 51F0
4FAB            DEFB    F0H             30, 51F0
4FAC            DEFB    F0H             30, 51F0
4FAD            DEFB    F0H             30, 51F0
4FAE            DEFB    F0H             30, 51F0
4FAF            DEFB    F0H             30, 51F0
4FB0            DEFB    F0H             30, 51F0
4FB1            DEFB    F0H             30, 51F0
4BB2            DEFB    F0H             30, 51F0
4FB3            DEFB    F0H             30, 51F0
4FB4            DEFB    F0H             30, 51F0
4FB5            DEFB    F0H             30, 51F0
4FB6            DEFB    F0H             30, 51F0
4FB7            DEFB    F0H             30, 51F0
4FB8            DEFB    F0H             30, 51F0
4FB9            DEFB    F0H             30, 51F0
4FBA            DEFB    F0H             30, 51F0
4FBB            DEFB    F0H             30, 51F0
4FBC            DEFB    F0H             30, 51F0
4FBD            DEFB    F0H             30, 51F0
4FBE            DEFB    F0H             30, 51F0
4FBF            DEFB    F0H             30, 51F0
4FC0            DEFB    F0H             30, 51F0
4FC1            DEFB    F0H             30, 51F0
4FC2            DEFB    F0H             30, 51F0
4FC3            DEFB    F0H             30, 51F0
*
*               Pnrase Address List
*
*                       Address         (Word)
*
4FC4            DEFW    5030H           (NO)
4FC6            DEFW    5032H           (ERROR)
4FC8            DEFW    5037H           (FORMAT)
4FCA            DEFW    503DH           (PARITY)
4FCC            DEFW    5043H           (DURING)
4FCE            DEFW    5049H           (HEADER)
4FD0            DEFW    504FH           (DATA)
4FD2            DEFW    5053H           (SEEK)
4FD4            DEFW    5057H           (ReAD))
4FD6            DEFW    505BH           (WRITE)
4FD8            DEFW    5060H           (LOST)
4FDA            DEFW    5064H           (NOT)
4FDC            DEFW    5067H           (ATTEMPTED TO)
4FDE            DEFW    5073H           (LOCKED/DELETED)
4FE0            DEFW    5081H           (SYSTEM)
4FE2            DEFW    5087H           (DIRECTORY)
4FE4            DEFW    5090H           (MEMORY)
4FE6            DEFW    5096H           (ON)
4FE8            DEFW    5098HH          (DISK)
4FEA            DEFW    509CH           (DISKETTE)
4FEC            DEFW    50A4H           (FAULT)
4FEE            DEFW    50A9H           (PROTECTED)
4FF0            DEFW    50B2H           (ILLEGAL)
4FF2            DEFW    50B9H           (LOGICAL)
4FF4            DEFW    50C0H           (NUMBER)
4FF6            DEFW    50C6H           (FILE)
4FF8            DEFW    50CAH           (RECORD)
4FFA            DEFW    50D0H           (END)
4FFC            DEFW    50D3H           (OF)
4FFE            DEFW    50D5H           (OUT)
5000            DEFW    50D8H           (RANGE)
5002            DEFW    50DDH           (ENCOUNTERED)
5004            DEFW    50E8H           (CODE)
5006            DeFW    50ECH           (GAT)
5008            DEFW    50EFH           (HIT)
500A            DEFW    50F2H           (OVERLAY)
500C            DEFW    50F9H           (UNKNOWN)
500E            DEFW    5100H           (LOAD)
5010            DEFW    5104H           (SPACE)
5012            DEFW    5109H           (ONLY)
5014            DEFW    510DH           (NAME)
5016            DEFW    5111H           (DEVICE)
5018            DEFW    5117H           (FORMAT)
501A            DEFW    511DH           (FOUND)
501C            DEFW    5122H           (IN)
501E            DEFW    5124H           (ACCESS)
5020            DEFW    512AH           (FULL)
5022            DEFW    512EH           (DRIVE)
5024            DEFW    5133H           (DENIED)
5026            DEFW    5139H           (PROGRAM)
5028            DEFW    5140H           (AVAILABLE)
502A            DEFW    5149H           (-CAN'T EXTEND)
502C            DEFW    5157H           (OPEN)
502E            DEFW    515BH           (WRITE)
*
*               Phrase List
*
5030            DEFM    'NO'                    (01)
5032            DEFM    'ERROR'                 (02)
5037            DEFM    'FORMAT'                (03)
503D            DEFM    'PARITY'                (04)
5043            DEFM    'DURING'                (05)
5049            DEFM    'HEADER'                (06)
504F            DEFM    'DATA'                  (07)
5053            DEFM    'SEEK'                  (08)
5057            DEFM    'READ'                  (09)
505B            DEFM    'WRITE'                 (10)
5060            DEFM    'LOST'                  (11)
5064            DEFM    'NOT'                   (12)
5067            DEFM    'ATTEMPTED TO'          (13)
5073            DEFM    'LOCKED'                (14)
5079            DEFM    '/DELETED'              (15)
5081            DEFM    'SYSTEM'                (16)
5087            DEFM    'DIRECTORY'             (17)
5090            DEFM    'MEMORY'                (18)
5096            DEFM    'ON'
5098            DEFM    'DISK'                  (19)
509C            DEFM    'DISKETTE'              (20)
50A4            DEFM    'FAULT'                 (21)
50A9            DEFM    'PROTECTED'             (22)
50B2            DEFM    'ILLEGAL'               (23)
50B9            DEFM    'LOGICAL'               (24)
50C0            DEFM    'NUMBER'                (25)
50C6            DEFM    'FILE'                  (26)
50CA            DEFM    'RECORD'                (27)
50D0            DEFM    'END'                   (28)
50D3            DEFM    'OF'                    (29)
50D5            DEFM    'OUT'                   (30)
50D8            DEFM    'RANGE'                 (31)
50DD            DEFM    'ENCOUNTERED'           (32)
50E8            DEFM    'CODE'                  (33)
50EC            DEFM    'GAT'                   (34)
50EF            DEFM    'HIT'                   (35)
50F2            DEFM    'OVERLAY'               (36)
50F9            DEFM    'UNKNOWN'               (37)
50FF            DEFM    'LOAD'                  (38)
5104            DEFM    'SPACE'                 (39)
5109            DEFM    'ONLY'                  (40)
510D            DEFM    'NAME'                  (41)
5111            DEFM    'DEVICE'                (42)
5117            DEFM    'FORMAT'                (43)
511D            DEFM    'FOUND'                 (44)
5122            DEFM    'IN'                    (45)
5124            DEFM    'ACCESS'                (46)
512A            DEFM    'FULL'                  (47)
512E            DEFM    'DRIVE'                 (48)
5133            DEFM    'DENIED'                (49)
5139            dEFM    'PROGRAM'               (50)
5140            DEFM    'AVAILABLE'             (51)
5149            DEFM    '- CAN'T  EXTENT'       (52)
5157            DEFM    'OPEN'                  (53)
*
*               Word Index List
*
515B            DEFB    01H,82H,06H,89H         NO ERROR
515E            DEFB    04H,02H,05H,06H,89H     PARITY ERROR DURING
                                                HEADER READ
5162            DEFB    08H,02H,05H,89H         SEEK ERROR DURING
                                                READ
5166            DEFB    0BH,07H,05H,89H         LOST DATA DURING
                                                READ
516A            DEFB    04H,02H,05H,89H         PARITY ERROR DURING
                                                READ
516E            DEFB    07H,1BH,0CH,2CH,05H,89H DATA RECORD NOT
                                                FOUND
                                                DURING READ
5174            DEFB    0DH,09H,0EH,07H,9BH     ATTEMPTED TO READ
                                                LOCKED RECORD
5179            DEFB    0DH,09H,0FH,07H,9BH     ATTEMPTED TO READ
                                                DELETED RECORD
517E            DEFB    2AH,0CH,F3H             DEVICE NOT AVAILABLE
5181            DEFB    04H,02H,05H,06H,8AH     PARITY ERROR DURING
                                                HEADER WRITE
5184            DEFB    08H,02H,05H,8AH         SEEK ERROR DURING
                                                WRITE
518A            DEFB    0BH,07H,05H,8AH         LOST DATA DURING
                                                WRITE
```

```
518E            DEFB    04H,02H,05H,8AH         PARITY ERROR DURING
                                                WRITE
5192            DEFB    07H,1BH,0CH,2CH,05H,8A  DATA RECORD NOT
                                                FOUND
                                                DURING WRITE
5198            DEFB    0AH,15H,12H,13H,B0H     WRITE FAULT MEMORY
                                                DISK DRIVE
519D            DEFB    0AH,16H,94H             WRITE PROTECTED
                                                DISKETTE
51A0            DEFB    17H,18H,1AH,99H         ILLEGAL LOGICAL FILE
                                                NUMBER
51A4            DEFB    10H,09H,82H             SYSTEM READ ERROR
51A7            DEFB    10H,0AH,82H             SYSTEM WRITE ERROR
51AA            DEFB    17H,1AH,E9H             ILLEGAL FILE NAME
51AD            DEFB    22H,09H,82H             GAT READ ERROR
51B0            DEFB    22H,0AH,82H             GAT WRITE ERROR
51B3            DEFB    23H,09H,82H             HIT READ ERROR
51B6            DEFB    23H,0AH,82H             HIT WRITE ERROR
51B9            DEFB    1AH,0CH,2DH,D0H         FILE NOT IN SYSTEM
51BD            DEFB    1AH,2EH,F1H             FILE ACCESS DENIED
51C0            DEFB    10H,27H,EFH             SYSTEM SPACE FULL
51C3            DEFB    13H,27H,AFH             DISK SPACE FULL
51C6            DEFB    1CH,1DH,1AH,A0H         END OF FILE
                                                ENCOuNTERED
51CA            DEFB    1B,19H,1EH,1DH,9FH      RECORD NUMBER OUT OF
                                                RANGE
51CF            DEFB    10H,2FH,34H,9AH         SYSTEM FULL CAN'T
                                                EXTEND FILE
51D6            DEFB    17H,30H,D9H             ILLEGAL DRIVE NUMBER
51D9            DEFB    01H,2AH,27H,F3H         NO DEVICE SPACE
                                                AVAILABLE
51DD            DEFB    26H,1AH,2BH,82H         LOAD FILE FORMAT
                                                ERROR
51E1            DEFB    11H,95H                 DIRECTORY FAULT
51E4            DEFB    0DH,26H,09H,28H,91H     ATTEMPTED TO READ
                                                DIRECTORY
51E8            DEFB    17H,2EH,0DH,16H,9AH     ILLEGAL ACCESS
                                                ATTEMPTED TO
                                                PROTECTED FILE
51ED            DEFB    1AH,0CH,F5H             FILE NOT OPEN
51F0            DEFB    25H,02H,A1H             UNKNOWN ERROR CODE
```

```
*************************************************************
*                                                           *
*                        SYS5/SYS                           *
*                     DEBUG Utility                         *
*                                                           *
*************************************************************


4E00 F1          POP     AF          Clear unnec. PUSH, restore AF
4E01 F1          POP     AF          Then create a 'hole' for SP
4E02 F5          PUSH    AF          Will be filled in at 4E1F
4E03 FDE5        PUSH    IY          Save IX, no alternate
4E05 DDE5        PUSH    IX          Save IY, no alternate
*
*               Save Context of Alternate Registers
*
4E07 08          EX      AF,AF'      Switch to alternate
4E08 D9          EXX                 Register set
4E09 E5          PUSH    HL          Save alternate HL
4E0A D5          PUSH    DE          Save alternatete DE
4E0B C5          PUSH    BC          Save alternate BC
4E0C F5          PUSH    AF          Save alternate AF
4E0D 08          EX      AF,AF'      Switch to primairy
4E0E D9          EXX                 Register set
4E0F E5          PUSH    HL          Save main HL
4E10 D5          PUSH    DE          Save main DE
4E11 C5          PUSH    BC          Save main BC
4E12 F5          PUSH    AF          Save main AF
4E13 210000      LD      HL,0000H    Clear HL so we can load SP into HL
4E16 39          ADD     HL,SP       HL points to start of saved context
4E17 116540      LD      DE,4065H    DEBUG context save area addr.
4E1A 011800      LD      BC,0018H    Number of bytes in saved context
4E1D EDB0        LDIR                Copy from stack to low mem. save area
4E1F 227940      LD      (4079H),HL  Save value of SP at time of trap
4E22 F9          LD      SP,HL       Restore SP value at time of trap
4E23 2A7B40      LD      HL,(407BH)  Addr of PC register +1 at trap time.
4E26 2B          DEC     HL          Back-up to addr. of trap instr.
                                     to get continue addr.
4E27 7E          LD      A,(HL)      Fetch instr. that caused trap
4E28 FEF7        CP      0F7H        Was it a RST 30 (trap instr.)
4E2A 2003        JR      NZ,4E2FH    Jump if not, may have been entered
                                     because of BREAK key hit
4E2C 227B40      LD      (407BH),HL  Yes, update PC reg. in context area
4E2F 215D40      LD      HL,405DH    Load trap address table
*
*       Trap address table contains original contents of
*       address where traps have been stored. Format of
*       entries is as follows:
*
*                        XX    LSB
*                        XX    MSB
*                        XX    Original value
*
*     Restore Original Contents of Locations That Have Traps
*
4E32 0602        LD      B,02H       .   Number of entries in table.
4E34 AF          XOR     A           .<- Zero A
4E35 0E02        LD      C,02H       .   Unnecessary instruction
4E37 5E          LD      E,(HL)      .   Get LSB of possible trap addr (TA)
4E38 77          LD      (HL),A      .   Zero LSB of TA from list
4E39 23          INC     HL          .   Bump to MSB of TA
4E3A 56          LD      D,(HL)      .   Fetch MSB of TA from list
4E3B 77          LD      (HL),A      .   Zero MSB of TA
4E3C 23          INC     HL          .   Bump to original value of TA
4E3D 7B          LD      A,E         .   Combine LSB and MSB
4E3E B2          OR      D           .   Of trap address
4E3F 2807        JR      Z,4E48H     .-  Jump if no entry, else check
                                     .|  for valid entry
4E41 1A          LD      A,(DE)      .|  Fetch contents of trap addr.
4E42 FEF7        CP      0F7H        .|  Is it a RST 30 (trap instruction)
4E44 2002        JR      NZ,4E48H    .-> No, check next entry
4E46 7E          LD      A,(HL)      .|  Yes, restore original instruction
                                     .|  Fetch saved instruction
4E47 12          LD      (DE),A      .|  and restore to location
                                     .|  that contains trap instruction
4E48 23          INC     HL          .<- Bump to next entry in list
4E49 10E9        DJNZ    4E34H       .-  Loop till all instr. restored
*
*                  Input Command Loop
*
4E4B ED7B7940    LD      SP,(4079H)  Restore context table value to SP
4E4F CDCF4E      CALL    4ECFH       Display register values (X display)
4E52 21C03F      LD      HL,3FC0H    Addr. of next to last row of display
4E55 222040      LD      (4020H),HL  Save as input line display area
4E58 CD8A51      CALL    518AH       Get a char. of input
4E5B FE47        CP      47H         Test for G
4E5D CA804F      JP      Z,4F80H     Jump if G command (Go command)
4E60 214B4E      LD      HL,4E4BH    Return addr. for all other commands
4E63 E5          PUSH    HL          To the stack
4E64 FE53        CP      53H         Test for S
4E66 2832        JR      Z,4E9AH     Jump if S command
4E68 FE3B        CP      3BH         Not S, test for;
4E6A 2842        JR      Z,4EAEH     Jump if semi-colon (increment
                                     memory display 1 page)
4E6C FE2D        CP      2DH         Not ";" test for "-"
4E6E 2856        JR      Z,4EC6H     Jump if dash (decrement memory
                                     display 1 page)
4E70 FE41        CP      41H         Not dash, test for A
4E72 2857        JR      Z,4ECBH     Jump if A command (ASCII display)
4E74 FE43        CP      43H         Not A, test for C
4E76 280A        JR      Z,4E82H     Go if C cmd (Single step with calls)
4E78 FE44        CP      44H         Not C, test for D
4E7A 282C        JR      Z,4EA8H     Jump if D command (Dislay memory)
4E7C FE48        CP      48H         Not D, test for H
4E7E 284B        JR      Z,4ECBH     Jump if H cmd (Hexadecimal display)
4E80 FE49        CP      49H         Not H, test for I and go if
4E82 CA5D50      JP      Z,505DH     I command (Single step next instr)
4E85 FE4D        CP      4DH         Not I, test for M (Enter
                                     memory modification mode)
4E87 CADB4F      JP      Z,4FDBH     Jump if M command
4E8A FE52        CP      52H         Not M, test for R (mod. reg. contents)
4E8C CA1150      JP      Z,5011H     Jump if R command
4E8F FE55        CP      55H         Not R, test for U (Dynamic display)
4E91 280B        JR      Z,4E9EH     Jump if U command (Dynamic display)
4E93 FE58        CP      58H         Not U, test for X (Return
                                     to primary display)
4E95 2802        JR      Z,4E99H     Jump if X command to
4E97 EF          RST     28H         Process input char as overlay request
4E98 C9          RET                 Then return to 4E4B
*
*            X Command. Return to Primary Display.
*
4E99 AF          XOR     A           Clear A-reg.
4E99 325E40      LD      (405EH),A   Signal short main display)
4E9D C9          RET                 Return to 4E4B
*
*              U Command. Dynamic Display
*
4E9E CD2B00      CALL    002BH       .<- Scan keyboard
4EA1 B7          OR      A           .|  Set flags for active key test
4EA2 C0          RET     NZ          .|  Ret to 4E4B if any key active
4EA3 CDCF4E      CALL    4ECFH       .|  Update display
4EA6 18F6        JR      4E9EH       .-  Loop till any key struck
*
*              D Command. Dislay Memory.
*
4EA8 CDA351      CALL    51A3H       Get beginning addr. from keyboard
4EAB C8          RET     Z           Rtn. to 4E4B if any error
4EAC 1814        JR      4EC2H       Else save starting display address
*
*   ; or - Command. Adjust Display Address (Forward or Backward)
*
4EAE 014000      LD      BC,0040H    Amt. to advance display (1 page);
4EB1 2A6340      LD      HL,(4063H)  Current beginning display address
4EB4 3A5E40      LD      A,(405EH)   Get display mode flag
4EB7 B7          OR      A           Set status for display mode
4EB8 2807        JR      Z,4EC1H     Jump if short display
4EBA 0E00        LD      C,00H       Clear LSB of adjustment address
4EBC 78          LD      A,B         Fetch MSB of adjustment addr and
4EBD B7          OR      A           Set flags for MSB of adjustment addr
4EBE 2001        JR      NZ,4EC1H    Jump if negative adjustment
4EC0 04          INC     B           Gives 100 (positive adjustment)
4EC1 09          ADD     HL,BC       Compute new display address, then
4EC2 226340      LD      (4063H),HL  save beginning display address
4EC5 C9          RET                 Ret to 4E4B
*
*            - Command. Decrement Memory Display
*
4EC6 01C0FF      LD      BC,0FFC0H   Adj. value for previous page
4EC9 18E6        JR      4EB1H       Go update display address
*
*                       A/H Command.
*
4ECB 325D40      LD      (405DH),A   Save display mode value
4ECE C9          RET                 Return to main display loop
*
*                       Register display
*
4ECF 3A5E40      LD      A,(405EH)   Get display mode flag
4ED2 B7          OR      A           Set status flags for display size
4ED3 2070        JR      NZ,4F45H    Jump if not register display
4ED5 3E1C        LD      A,1CH       Home cursor command
4ED7 CD3300      CALL    0033H       Give video driver home cursor command
4EDA 216540      LD      HL,4065H    Address of saved register context
4EDD E5          PUSH    HL          Save context address
4EDE 21544F      LD      HL,4F54H    Address of register name text
4EE1 060C        LD      B,0CH       No. of register values to
display
4EE3 1805        JR      4EEAH       Go display register context
*
*       Display register values plus the contents of the
*            addresses contained therein.
*
4EE5 3E0D        LD      A,0DH       .   Use a carriage return to
4EE7 CD3300      CALL    0033H       .   Skip to next line of display
4EEA CDF651      CALL    51F6H       .   Display reg. name pointed to by HL
4EED E3          EX      (SP),HL     .   Addr. of next reg. name
                                     .   to stack. Context addr. for
4EEE 5E          LD      E,(HL)      .<- Register to HL. Load LSB of
                                     .|  reg. value from context list
4EEF 23          INC     HL          .|  Bump to MSB of context list
4EF0 56          LD      D,(HL)      .|  Fetch MSB of reg. value
4EF1 23          INC     HL          .|  Bump to addr. of next reg. set
4EF2 E5          PUSH    HL          .|  Save addr. in context list
4EF3 EB          EX      DE,HL       .|  Value for that reg. to HL
4EF4 3E3D        LD      A,3DH       .|  ASCII value for =
4EF6 CD3300      CALL    0033H       .|  Display =
4EF9 CDF251      CALL    51F2H       .|  Display a blank following =
4EFC 7C          LD      A,H         .|  A = MSB of context value
4EFD CDEF51      CALL    51EFH       .|  Convert to ASCII hex and display
4F00 7D          LD      A,L         .|  A = LSB of context value
4F01 CDEF51      CALL    51EFH       .|  Convert to ASCII hex and display
4F04 78          LD      A,B         .|  A = remaining number of register
                                     .|  values to be displayed
4F05 E60B        AND     0BH         .|  Test if index is exactly 8
4F07 FE08        CP      08H         .|  and if not, display status flags
4F09 201C        JR      NZ,4F27H    .-  Jump if not displaying AF reg.
4F0B 4D          LD      C,L         .|  Status flags to C register
4F0C C5          PUSH    BC          .|  Save no. of reg. to be displayed
4F0D 21784F      LD      HL,4F78H    .|  Addr of ASCII chars for status bits
4F10 0608        LD      B,08H       .<- B = no. of stat bits to examine
4F12 CB21        SLA     C           .|  Shift a status bit into CARRY
4F14 7E          LD      A,(HL)      .|  Get letter for status condition
4F15 3802        JR      C,4F19H     .-  Jump if status bit is on
                                     .|  and display letter, else
4F17 3E2D        LD      A,2DH       .|  ASCII value for -
4F19 CD3300      CALL    0033H       .<- Display a - (or letter if
                                     .|  condition is true)
```

```
4F1C 23          INC     HL          .  |  Bump to next letter in status list
4F1D 10F3        DJNZ    4F12H       .  |  Loop till all status bits
                                     .  |  tested and displayed
4F1F C1          POP     BC          .<-|  restore no. reg. left to display
4F20 3EEC        LD      A,0ECH      .  |  Comparison code for 44 blanks
4F22 CD3300      CALL    0033H       .  |  Display 44 blanks
4F25 1803        JR      4F2AH       .  |  Skip over => value display
4F27 CD3151      CALL    5131H       .--|  Display => followed by
                                     .<-|  contents of addr. in reg.
4F2A E1          POP     HL          .  |  Restore addr next register set
4F2B E3          EX      (SP),HL     .  |  Put address on stack
4F2C 10B7        DJNZ    4EE5H       .--|  Loop till all registers displayed
4F2E E1          POP     HL             Clear stack
4F2F 2A6340      LD      HL,(4063H)     Load beginning display address
*
*      Display memory starting at address specified in HL.
*
4F32 0604        LD      B,04H          B = number of lines to display
4F34 3EC7        LD      A,0C7H      .<-Comparison code for 7 blanks
4F36 CD3300      CALL    0033H       .  |  Display 7 blanks
4F39 CDD451      CALL    51D4H       .  |  Display hex. for value in HL
                                     .  |  register (address)
4F3C CDF251      CALL    51F2H       .  |  Display a blank
4F3F CD3151      CALL    5131H       .  |  Display 16 bytes
                                     .  |  starting at address
4F42 10F0        DJNZ    4F34H       .--|  Loop till (B) lines displayed
4F44 C9          RET                 .     Return to caller
*
*                    Start of Long Display
*
4F45 21FF3B      LD      HL,3BFFH       Address -1 of video display area
4F48 222040      LD      (4020H),HL     to next char. address in video DCB
4F4B 2A6340      LD      HL,(4063H)     Fetch display address and
4F4E 2E00        LD      L,00H          force it to a page boundary
4F50 0610        LD      B,10H          B = number of lines to display
4F52 18E0        JR      4F34H          Go display 16 lines of memory
*
*                    Text String for X-display
*
4F54             DEFM    'AF '
4F57             DEFM    'BC '
4F5A             DEFM    'DE '
4F5D             DEFM    'HL '
4F60             DEFM    'AF''
4F63             DEFM    'BC''
4F66             DEFM    'DE''
4F69             DEFM    'HL''
4F6C             DEFM    'IX '
4F6F             DEFM    'IY '
4F72             DEFM    'SP '
4F75             DEFM    'PC '
4F78             DEFM    'SZ1H1PNC'
*
*                              G Command
*
4F80 0602        LD      B,02H          Max. no. trap addr to input
4F82 116240      LD      DE,4062H       Address of trap table
4F85 CDA351      CALL    51A3H          Input a hex. value (1st address)
4F88 2803        JR      Z,4F8DH        Jump if illegal char. detected
4F8A 227B40      LD      (407BH),HL     Save new PC (continue address)
4F8D 380A        JR      C,4F99H     .<-Jump if no comma after 1st address
4F8F CDA351      CALL    51A3H       .  |  Get a hex.value (2nd address)
4F92 F5          PUSH    AF          .  |  Preserve status
4F93 C4CA4F      CALL    NZ,4FCAH    .  |  Call routine to update trap
                                     .  |  table addr. if legit. value
4F96 F1          POP     AF          .  |  Restore status 1st addr read
4F97 10F4        DJNZ    4F8DH       .--|  Loop till 2 addr or car. ret found
4F99 210F40      LD      HL,400FH       Vector address to load DEBUG
4F9C 221643      LD      (4316H),HL     put in nucleus
4F9F 3EC3        LD      A,0C3H         OP code for JUMP instruction
4FA1 321543      LD      (4315H),A      put in nucleus
4FA4 217A40      LD      HL,407AH       Addr of context starting with SP
4FA7 060B        LD      B,0BH          No. of reg. context values to transfer
*
*      Transfer register values from context area to current stack.
*
4FA9 56          LD      D,(HL)      .<-Fetch MSB of saved value
4FAA 2B          DEC     HL          .  |  Skip back to LSB
4FAB 5E          LD      E,(HL)      .  |  Fetch LSB of saved register value
4FAC 2B          DEC     HL          .  |  Skip to next register value
4FAD D5          PUSH    DE          .  |  Save value on current stack
4FAE 10F9        DJNZ    4FA9H       .--|  Loop till all values on stack
*
*                    Restore Register Context
*
4FB0 F1          POP     AF             Restore AF main register
4FB1 C1          POP     BC             Restore BC main register
4FB2 D1          POP     DE             Restore DE main register
4FB3 E1          POP     HL             Restore HL main register
4FB4 08          EX      AF,AF'         Switch to alternate AF register
4FB5 D9          EXX                    Switch to alternate register set
4FB6 F1          POP     AF             Restore AF alternate register
4FB7 C1          POP     BC             Restore BC alternate register
4FB8 D1          POP     DE             Restore DE alternate register
4FB9 E1          POP     HL             Restore HL alternate register
4FBA 08          EX      AF,AF'         Switch to main AF register
4FBB D9          EXX                    Switch to main register
4FBC DDE1        POP     IX             Restore IX
4FBE FDE1        POP     IY             and IY
4FC0 E1          POP     HL             and stack pointer at trap time
4FC1 F9          LD      SP,HL          Update SP to value at time of trap
4FC2 2A7B40      LD      HL,(407BH)     Resumption address to HL
4FC5 E5          PUSH    HL             So it can be copied to stack
4FC6 2A6B40      LD      HL,(406BH)     Restore HL main register set
4FC9 C9          RET                    Return to inst. where trap occurred
*
*    Stuff trap inst. at addr. in HL. Save original contents and
*    addr. of contents at list addr. in DE.
*
4FCA 7E          LD      A,(HL)         Get original conts. of trap location.
4FCB 12          LD      (DE),A         and save in trap table list
4FCC 1B          DEC     DE             Back-up to next entry in trap list
4FCD 3EF7        LD      A,0F7H         Load a trap inst. RST 30
4FCF 77          LD      (HL),A         Save at addr. in HL, then
4FD0 BE          CP      (HL)           Make sure store worked
4FD1 C22F4E      JP      NZ,4E2FH       Jump if store inst. failed
4FD4 7C          LD      A,H            MSB of trap addr.
4FD5 12          LD      (DE),A         To trap table
4FD6 1B          DEC     DE             Backspace trap table addr.
4FF7 7D          LD      A,L            LSB of trap addr.
4FD8 12          LD      (DE),A         To trap table
4FD9 1B          DEC     DE             Back-up to 1st byte of next entry
4FDA C9          RET                    Return to caller
*
*                              M Command
*
4FDB 2A6040      LD      HL,(4060H)     Current modify address
4FDE CDA351      CALL    51A3H          Get next modify address
4FE1 226040      LD      (4060H),HL     Save previous, or new addr if entered
4FE4 D8          RET     C              Ret to caller if carriage return
4FE5 E5          PUSH    HL             Save addr of location to be modified
4FE6 CDCF4E      CALL    4ECFH          Update display in case it's changed
4FE9 21403F      LD      HL,3F40H       Screen addr for current mem contents
4FEC 222040      LD      (4020H),HL     Save as cursor addr in video DCB
4FEF 2A6040      LD      HL,(4060H)     Modify address to HL
4FF2 CDD451      CALL    51D4H          Display address to be modified
4FF5 E5          PUSH    HL             Save addr of location to modify
4FF6 21803F      LD      HL,3F80H       Get addr to display modified mem and
4FF9 222040      LD      (4020H),HL     save as new cursor addr in video DCB
4FFC E1          POP     HL             Restore addr of location to modify
4FFD CDD051      CALL    51D0H          Display conts. of location to be
                                        modified. Follow it with a hyphen "-"
5000 3E2D        LD      A,2DH          ASCII value for -
5002 CD3300      CALL    0033H          Display -
5005 D1          POP     DE             Reload addr of location to modify
5006 CDA351      CALL    51A3H          Input new value for this location
5009 EB          EX      DE,HL          Save it in DE. Storage addr to HL
500A 801         JR      Z,500DH        If space input don't modify,
500C 73          LD      (HL),E         save new value
500D D8          RET     C              Exit if carriage return entered
500E 23          INC     HL             else bump to next address
500F 18D0        JR      4FE1H          Go update display, get replacement val.
*
*                              R Command
*
5011 CD8A51      CALL    518AH          Get 1st letter of reg pair (next char)
5014 C8          RET     Z              Ret if comma, blank or C.R. (error)
5015 4F          LD      C,A            Save 1st letter of register pair
5016 CD8A51      CALL    518AH          Get 2nd letter of reg pair (next char)
5019 C8          RET     Z              Ret if comma, blank or C.R. (error)
501A 57          LD      D,A            Save 2nd letter of register pair
501B 1E20        LD      E,20H          Default RP terminator. Use for comma
501D CD8A51      CALL    518AH          Get char. after register pair.
                                        Should be blank or ","
5020 D8          RET     C              Ret if control. Error if not blank
5021 2806        JR      Z,5029H        Go if comma/blank (legal separator)
5023 5F          LD      E,A            Save actual char. Could be '
5024 CD8A51      CALL    518AH          Get next char. Must be comma/blank
5027 C0          RET     NZ             Exit if
5028 D8          RET     C              neither. Ignore illegal command
5029 21544F      LD      HL,4F54H       Address of register pair table
502C 060C        LD      B,0CH          Number of pairs in table. (12)
502E 7E          LD      A,(HL)      .<-   Get first char. of register pair
502F B9          CP      C           .  |  Compare to first char.
                                     .  |  of pair entered
5030 2806        JR      Z,5038H     .  |  If so, go test 2nd char
5032 23          INC     HL          .  |  Else skip over 2nd
5033 23          INC     HL          .  |  and 3rd char to 1st
5034 23          INC     HL          .<-|  char of next entry
5035 10F7        DJNZ    502EH       .--|  Loop till all entries checked
5037 C9          RET                 .     Ignore cmd if no match in list
5038 23          INC     HL          .<-   Bump to 2nd char of RP name
5039 7E          LD      A,(HL)      .     Get 2nd char in RP name list
503A BA          CP      D           .     Compare to 2nd char.
                                     .     of name entered
503B 20F6        JR      NZ,5033H    .     Go if no match. Scan rest of list
503D 23          INC     HL          .     Bump to 3rd char in RP name list
503E 7E          LD      A,(HL)      .     Get it. Will be blank or quote
503F BB          CP      E           .     Compare to value entered
5040 20F2        JR      NZ,5034H    .--   Go if no match (input error)
5042 3E18        LD      A,18H          2 times Max no. of entries in RP list
5044 90          SUB     B              Minus offset to
5045 90          SUB     B              matching entry gives
5046 4F          LD      C,A            byte offset into
5047 0600        LD      B,00H          saved register pair area
5049 216540      LD      HL,4065H       Origin of save register pair
                                        values in low memory
504C 09          ADD     HL,BC          Compute addr of RP to be modified
504D E5          PUSH    HL             Save addr of RP to be altered
504E 3E1E        LD      A,1EH          Code to erase to end of line
5050 CD3300      CALL    0033H          Blank remainder of line
5053 D1          POP     DE             DE = address of RP to change
5054 CDA351      CALL    51A3H          Put value for register pair in DE
5057 C8          RET     Z              Exit if illegal character entered
5058 EB          EX      DE,HL          Value to DE. Address to HL
5059 73          LD      (HL),E         Save LSB of new register value
505A 23          INC     HL             Bump to next byte
505B 72          LD      (HL),D         Save MSB of new register value
505C C9          RET                    Return to caller
```

```
*
*                Single Step (I and C Commands)
*
505D F5          PUSH    AF              Save command specification
505E ED5B7B40    LD      DE,(407BH)      Load resumption address
5062 1A          LD      A,(DE)          Load OP code at 'resume' address
5063 211551      LD      HL,5115H        Addr of group 3 type instructions
5066 FEDD        CP      0DDH            Test for indexed instruction
5068 280E        JR      Z,5078H         Jump if indexed IX instr.
506A FEFD        CP      0FDH            If not, test for IY indexed instr.
506C 280A        JR      Z,5078H         Jump if indexed IY
506E 21E050      LD      HL,50E0H        Addr of group 1 type instructions
5071 FEED        CP      0EDH            Test for special instruction
5073 2006        JR      NZ,507BH        Go if not indexed or special instr.
5075 21QE51      LD      HL,510EH        Addr of group 2 type instructions
5078 13          INC     DE              Bump to next byte of instruction
5079 1A          LD      A,(DE)          Fetch next instruction byte
507A 1B          DEC     DE              Backspace to start of instruction
507B 4F          LD      C,A             C = LSB of OP code
507C 7E          LD      A,(HL)          .<-Fetch OP code mask
507D A1          AND     C               .  Apply mask to LSB of OP code
507E 23          INC     HL              .  Bump to OP code value in
                                         .  instruction list
507F BE          CP      (HL)            .  Compare to OP code in table
5080 23          INC     HL              .-Bump to code length in list
5081 2806        JR      Z,5089H         .  Jump if OP codes match, else
5083 23          INC     HL              .  bump to next 3 byte entry
5084 7E          LD      A,(HL)          .  Fetch mask
5085 FE05        CP      05H             .-and test for end of list
5087 30F3        JR      NC,507CH        .<- Go if end of list not reached
5089 7E          LD      A,(HL)          Get code/length value for instruction
508A 47          LD      B,A             Save in B
508B E60F        AND     0FH             Isolate length of instruction to
                                         be single cycled
508D 6F          LD      L,A             Then form length of instruction
508E 2600        LD      H,00H           in HL, and
5090 19          ADD     HL,DE           add to addr of instr. to execute
5091 D5          PUSH    DE              Save address of next instruction
5092 116240      LD      DE,4062H        Load trap address save location
5095 CDCA4F      CALL    4FCAH           Original instr to 4062. Trap to (HL)
5098 E1          POP     HL              Restore addr of instr after current
5099 78          LD      A,B             Reload code/length for current instr
509A E6F0        AND     0F0H            Isolate type e.g. RET, JP, CALL ...
509C 2816        JR      Z,50B4H         Go if type 0. Exec. cont. at next addr.
509E 23          INC     HL              Bump addr. of next instr. by one
509F FE20        CP      20H             Test for JP type instr.
50A1 3835        JR      C,50D8H         Jump if JP (HL) instr. (Type 1)
50A3 2827        JR      Z,50CCH         Jump if JP (IX/IY) instr. (Type 2)
50A5 FE40        CP      40H             Test for JP  XXXX type instr.
50A7 3817        JR      C,50C0H         Jump if DJNZ (Type 3)
50A9 280F        JR      Z,50BAH         Jump if JP  XXXX (Type 4)
50AB FE60        CP      60H             Test for type 6
50AD 3808        JR      C,50B7H         Jump if code 5 (RET type instr.)
50AF F1          POP     AF              Restore command (I or C)
50B0 FE49        CP      49H             Compare with I. If so go
50B2 2806        JR      Z,50BAH         store trap at CALL addr and execute
50B4 C3A44F      JP      4FA4H           C command. Go restore context
                                         and execute instruction
50B7 2A7940      LD      HL,(4079H)      Load ret addr from context stack
50BA 7E          LD      A,(HL)          LSB of return addr.
50BB 23          INC     HL              Bump to MSB of ret.addr.
50BC 66          LD      H,(HL)          load MSB of return addr.
50BD 6F          LD      L,A             HL = return address
50BE 181B        JR      50DBH           Go restore context and exec. instr.
50C0 4E          LD      C,(HL)          Fetch offset for DJNZ, JR
50C1 AF          XOR     A               Clear status flags
50C2 CB79        BIT     07H,C           Test sign bit of offset displacement
50C4 2801        JR      Z,50C7H         Jump if positive. Forward jump
50C6 2F          CPL                     Negative displacement. Convert to
50C7 47          LD      B,A             8 bit negative value
50C8 23          INC     HL              Bump to addr of following instr.
50C9 09          ADD     HL,BC           Add offset to get addr of next instr.
50CA 180F        JR      50DBH           Go restore context, execute instr.
50CC 2A7540      LD      HL,(4075H)      Load IX in case of JP (IX)
50CF CB69        BIT     05H,C           Test if JP (IX)
50D1 2808        JR      Z,50DBH         If JP (IX) go execute instruction
50D3 2A7740      LD      HL,(4077H)      Load IY. JP (IY) instr.
50D6 1803        JR      50DBH           Go restore context, execute instr.
50D8 2A6B40      LD      HL,(406BH)      Get jump address from HL
50DB CDCA4F      CALL    4FCAH           Stuff trap instruction there
50DE 18D4        JR      50B4H           Go restore context, execute instr.
*
*          Group 1 INSTRUCTION LIST
*          Each entry has the following format:
*
*               DEFB    MASK, OP CODE, TYPE/LENGTH
*
*          TYPE = A code indicating where the address of the
*          next instruction may be found, type codes defined
*          as follows:
*          0 - Next instruction follows current instr.
*          1 - Next instr. addr. comes from HL as in JP (HL).
*          2 - Next instr. addr. conditionally comes from
*              operand, as in JP  C,XXXX.
*          3 - Conditional as in DJNZ  XXXX.
*          4 - Operand as in JP XXXX.
*          5 - Conditional and indirect as in RET, or RET NZ.
*          6 - Conditional, operand as in CALL  NZ,XXXX or
*              CALL  XXXX
*
50E0             DEFB    C7H,C0H,51H
50E3             DEFB    FH,C9H,51H
50E6             DEFB    FFH,E9H,11H
50E9             DEFB    CFH,01H,03H
50EC             DEFB    E7H,22H,03H
50EF             DEFB    C7H,C2H,43H
50F1             DEFB    FFH,C3H,43H
50F5             DEFB    C7H,C4H,63H
50F8             DEFB    FFH,CDH,63H
50FB             DEFB    C7H,06H,02H
50FE             DEFB    F7H,D3H,02H
5101             DEFB    C7H,C6H,02H
5104             DEFB    FFH,CBH,02H
5107             DEFB    F7H,10H,32H
510A             DEFB    E7H,20H,32H
510D             DEFB    01H
*
*   Default length for any instructions not matching those in list
*
*               GROUP 2 INSTRUCTION LIST
*               Each bythe has the following format
*               DEFB MASK, OP CODE, TYPE/LENGTH
*               LENGTH = length of the current op code
*
510E             DEFB    C7H,43H,04H
5111             DEFB    F7H,45H,52H
5114             DEFB    02H
*
* Default length for any instructions not matching those in above list
*
*          GROUP 3 INSTRUCTION LIST
*          Each byte has the following format:
*          DEFB  MASK, OP CODE, TYPE/LENGTH
*          LENGTH = length of the current op code
*
5115             DEFB    FEH,34H,03H
5118             DEFB    C0H,40H,03H
511B             DEFB    C0H,80H,03H
511E             DEFB    FFH,21H,02H
5122             DEFB    FFH,22H,04H
5124             DEFB    FFH,2AH,04H
5127             DEFB    FFH,36H,04H
512A             DEFB    FFH,CBH,04H
512D             DEFB    FFH,E9H,22H
5130             DEFB    02H
*
*          GROUP 3 INSTRUCTION LIST
*          Each byte has the following format
*          DEFB  MASK, OP CODE, TYPE/LENGTH
*          LENGTH = Length of the current op code
*
5131 C5          PUSH    BC              Preserve caller's BC
5132 3E3D        LD      A,3DH           ASCII value for =
5134 CD3300      CALL    0033H           Display =
5137 3E3E        LD      A,3EH           ASCII value for >
5139 CD3300      CALL    0033H           Display >
513C CDF251      CALL    51F2H           Display a space
513F 0610        LD      B,10H           Number of values to display.
5141 CD6551      CALL    5165H           If addr to modify is being
                                         displayed bracket it with bars
5144 3A5D40      LD      A,(405DH)       Fetch display mode flag
5147 FE41        CP      41H             Test for A
5149 2013        JR      NZ,515EH        Jump if not alphabetical
514B CDF251      CALL    51F2H           Display a blank
514E 7E          LD      A,(HL)          Fetch a character
514F FE20        CP      20H             Test for control code
                                         (compare to space)
5151 3804        JR      C,5157H         Jump if control code. Display a .
5153 FEC0        CP      0C0H            Test for compression code
5155 3802        JR      C,5159H         Jump if not a compression code
5157 3E2E        LD      A,2EH           ASCII value for .
5159 CD3300      CALL    0033H           Display character or .
515C 23          INC     HL              Bump to next character in list
515D AF          XOR     A               Clear status to skip CALL below
515E C4D051      CALL    NZ,51D0H        Display a hex value
5161 10DE        DJNZ    5141H           Loop till 16 values displayed
5163 C1          POP     BC              Restore caller's BC
5164 C9          RET                     Return to caller
*
*       Put Display Bars Around Location Being Modified
*
5165 ED5B6040    LD      DE,(4060H)      Load 2nd trap address into DE
5169 13          INC     DE              Bump to addr of following instruction
516A E5          PUSH    HL              Save current display address
516B AF          XOR     A               Clear CARRY for SBC operation
516C ED52        SBC     HL,DE           Compare display and modify address
516E 3E95        LD      A,95H           ASCII code for "left bar"
5170 280E        JR      Z,5180H         If display = modify addr.
                                         disp. left bar
5172 CD8451      CALL    5184H           If 8 values disp. add an extra space
5175 23          INC     HL              Bump display address
5176 7D          LD      A,L             Then combine
5177 B4          OR      H               LSB and MSB
5178 E1          POP     HL              Restore current display address
5179 3EAA        LD      A,0AAH          ASCII code for right bar
517B CA3300      JP      Z,0033H         If disp. addr = modify address
                                         Display closing right bar
517E 1808        JR      5188H           Display space between values and ret
5180 CD3300      CALL    0033H           Display left bar
5183 E1          POP     HL              Restore display address
5184 78          LD      A,B             Get no. of lines still to display
5185 FE08        CP      08H             If half a line (8) left, add a space
5187 C0          RET     NZ              Exit if not half way point yet
5188 1868        JR      51F2H           Go display a blank and ret to caller
*
*                  Get next character from keybaord
*
518A D5          PUSH    DE              Preserve callers DE
518B CD4900      CALL    0049H           .  Wait for next character
518E FE0D        CP      0DH             .  Was it a carriage ret.
5190 280E        JR      Z,51A0H         .  Jump if yes - end of line
5192 FE20        CP      20H             .  Not a carriage return, was it
                                         .  a control character
5194 38F5        JR      C,518BH         .  Jump if yes - ignore control codes
5196 CD3300      CALL    0033H           Else echo character received
                                         (display it)
5199 D1          POP     DE              Restore caller's DE
519A FE2C        CP      2CH             Was character received a comma?
519C C8          RET     Z               Return to caller if yes.
                                         Process this field
```

```
519D FE20       CP      20H         Not a comma, compare it with a blank
519F C9         RET                 and return to caller
51A0 D1         POP     DE          Restore caller's DE
51A1 37         SCF                 Signal end of line received
51A2 C9         RET                 and return to caller
*
*                       Input a hex value
*
51A3 CD8A51     CALL    518AH       Wait for next character
51A6 C8         RET     Z           Ret to caller if it's blank or comma
51A7 210000     LD      HL,0000H    Zero accum. to contain hex val on exit
51AA CDBF51     CALL    51BFH       Get binary value if it was a hex digit
51AD DA4B4E     JP      C,4E4BH     Jump if not a hex digit (0-9,A-F)
51B0 29         ADD     HL,HL       Digit times 2
51B1 29         ADD     HL,HL       Digit times 4
51B2 29         ADD     HL,HL       Digit times 8
51B3 29         ADD     HL,HL       Digit times 16. In case more
51B4 B5         OR      L           than 2 digits are entered
51B5 6F         LD      L,A         Only last 2 will be used
51B6 CD8A51     CALL    518AH       Get next digit and if not a blank
51B9 20EF       JR      NZ,51AAH    go test char, accumulate hex value
51BB 1F         RRA                 CARRY flag to Bit 7
51BC CE81       ADC     A,81H       Will set CARRY if carriage return
51BE C9         RET                 Return to caller
*
*       Test character. Return with CARRY if not 0 - 9, or A - F
*                       (hex digit)
*
51BF D630       SUB     30H         Compare with ASCII zero
51C1 D8         RET     C           Ret if less than 0. Obviously not hex
51C2 C6E9       ADD     A,0E9H      Test for a letter > F
51C4 D8         RET     C           Return if > F. Not a hex digit
51C5 C606       ADD     A,06H       Test for letter (A-Z)
51C7 3803       JR      C,51CCH     Jump if letter
51C9 C607       ADD     A,07H       Not letter, digit 0-9
51CB D8         RET     C           Unnecessary instruction
51CC C60A       ADD     A,0AH       Compute final binary value
51CE B7         OR      A           Set status flags for value
51CF C9         RET                 Return to caller
51D0 7E         LD      A,(HL)      Fetch an 8-bit hex value
51D1 23         INC     HL          Bump to address of next 8-bit value
51D2 1805       JR      51D9H       Go convert hex to ASCII and ret
51D4 7C         LD      A,H         MSB of value to convert and display
51D5 CDD951     CALL    51D9H       Display 2 hex digits (MSB), then
51D8 7D         LD      A,L         get LSB and display other 2 hex digits
51D9 F5         PUSH    AF          Save value to be displayed
51DA 1F         RRA                 Right shift value
51DB 1F         RRA                 four
51DC 1F         RRA                 places
51DD 1F         RRA                 to isolate and convert upper
51DE CDE251     CALL    51E2H       4 bits to hex ASCII and display
51E1 F1         POP     AF          Restore value being displayed
51E2 E60F       AND     0FH         Isolate lower 4 bits of value
51E4 C630       ADD     A,30H       Gives ASCII value for digit
51E6 FE3A       CP      3AH         Test for hex value A-F
51E8 3802       JR      C,51ECH     Jump if digit 0-9 else
51EA C607       ADD     A,07H       get ASCII value for A-F
51EC C33300     JP      0033H       .   Display char in A and return
51EF CDD951     CALL    51D9H       .   Display 2 hex digits
51F2 3E20       LD      A,20H       .   ASCII code for a blank
51F4 18F6       JR      51ECH       .   Display blank and return
51F6 CDFC51     CALL    51FCH       .   Get char from list in HL
                                    .   Display char and ret to 51F9
51F9 CDFC51     CALL    51FCH       .   Get next char from list
                                    .   Display and Ret to 51FC
51FC 7E         LD      A,(HL)      .   Get char from list addr in HL
51FD 23         INC     HL          .   Bump to next char. in list
51FE 18EC       JR      51ECH       .   Display char, ret.
                                    .   to caller of 51FC
```

```
*************************************************************
*                                                           *
*                        SYS6/SYS                           *
*       Command Processor for:                              *
*                                                           *
*       APPEND      ATTRIB      AUTO        CLOCK           *
*       COPY        DATE        DEVICE      DIR             *
*       DUMP        FREE        KILL        LIB             *
*       LIST        LOAD        PRINT       PROT            *
*       RENAME      TIME        VERIFY                      *
*                                                           *
*************************************************************


5200 79         LD      A,C         Get command index computed by SYS1
5201 012D40     LD      BC,402DH    Put a return address of 402D
5204 C5         PUSH    BC          onto the stack
5205 3D         DEC     A           Test if command index = 1
5206 CA0057     JP      Z,5700H     Jump if APPEND command
5209 3D         DEC     A           Test if command index = 2
520A CA005B     JP      Z,5B00H     Jump if ATTRIB command
520D 3D         DEC     A           Test if command index = 3
520E CA8352     JP      Z,5283H     Jump if AUTO command
5211 3D         DEC     A           Test if command index = 4
5212 CAC552     JP      Z,52C5H     Jump if CLOCK command
5215 3D         DEC     A           Test of command index = 5
5216 CA4F57     JP      Z,574FH     Jump if COPY command
5219 3D         DEC     A           Test if command index = 6
521A CA9D52     JP      Z,529DH     Jump if DATE command
521D 3D         DEC     A           Test if command index = 7
521E CA1D53     JP      Z,531DH     Jump if DEVICE command
5221 3D         DEC     A           Test if command index = 8
5222 CA6F5C     JP      Z,5C6FH     Jump if DIR command
5225 3D         DEC     A           Test if command index = 9
5226 CAD7557    JP      Z,57D7H     Jump if DUMP command
5229 3D         DEC     A           Test if command index = 10
522A CA205E     JP      Z,5E20H     Jump if FREE command
522D 3D         DEC     A           Test if command index = 11
522E CA1459     JP      Z,5914H     Jump if KILL command
5231 3D         DEC     A           Test if command index = 12
5232 CA3A53     JP      Z,533AH     Jump if LIB command
5235 3D         DEC     A           Test if command index = 13
5236 CA2C59     JP      Z,592CH     Jump if LIST command
5239 3D         DEC     A           Test if command index = 14
523A CA9459     JP      Z,5994H     Jump if LOAD command
523D 3D         DEC     A           Test if command index = 15
523E CAAC59     JP      Z,59ACH     Jump if PRINT command
5241 3D         DEC     A           Test if command index = 16
5242 CA6A53     JP      Z,536AH     Jump if PROT command
5245 3D         DEC     A           Test if command index = 17
5246 CA9C5F     JP      Z,5F9CH     Jump if RENAME command
5249 3D         DEC     A           Test if command index = 18
524A CAB152     JP      Z,52B1H     Jump if TIME command
524D 3D         DEC     A           Test if command index = 19
524E CA0455     JP      Z,5504H     Jump if VERIFY command
*
*           Command Index Out of Range
*
5251 215A52     LD      HL,525AH    Addr of RESERVED COMMAND message
5254 CD6744     CALL    4467H       Display message, then
5257 C33040     JP      4030H       Recall SYS1, option 2
525A            DEFM                'RESERVED COMMAND'
526A            DEFB    0DH         MESSAGE TERMINATOR
*
*
*
526B 217452     LD      HL,5274H    Address of NOTHING DONE message
526E CD6744     CALL    4467H       Display message
5271 C33040     JP      4030H       Return control to SYS1
                                    to get next command
5274            DEFM                'NOTHING DONE'
5281            DEFB    2DH
5282            DEFB    0DH         MESSAGE TERMINATOR
*
*               AUTO Command Processing
*
5283 0E00       LD      C,00H       Specify drive 0
5285 CDF04A     CALL    4AF0H       Get track 11, sector 0 into 4D00
5288 C24C55     JP      NZ,554CH    Jump if error while reading GAT
528B 164D       LD      D,4DH       Form addr of AUTO procedure in
528D 1E00       LD      E,00H       GAT sector buffer in DE (4D00)
528F 012000     LD      BC,0020H    No. of chars to put in AUTO area
5292 EDB0       LDIR                Copy 32 chars following AUTO command
                                    to procedure file name field
5294 0E00       LD      C,00H       Select drive 0
5296 CD034B     CALL    4B03H       Write updated GAT sector
5299 C24C55     JP      NZ,554CH    Go if error during write
529C C9         RET                 Return to SYS1
*
*                       DATE Routine
*
529D 0E2F       LD      C,2FH       / = Only valid terminating character
529F CDEA52     CALL    52EAH       Process date. Leave in 52DC-52DE
52A2 C2D352     JP      NZ,52D3H    Go if char not a number or /
52A5 21DC52     LD      HL,52DCH    Start of 3 byte date buffer
52A8 114440     LD      DE,4044H    Address of date in system area
52AB 010300     LD      BC,0003H    Number of bytes to move
52AE EDB0       LDIR                Copy date from buffer to system area
52B0 C9         RET                 Ret to SYS1 to get next command
*
*                       TIME Routine
*
52B1 0E3A       LD      C,3AH       C = : (only valid terminator)
52B3 CDEA52     CALL    52EAH       Process time. Leave in 52DC-52DE
52B6 C2D352     JP      NZ,52D3H    Go if anything but number or :
52B9 21DC52     LD      HL,52DCH    Addr of local buffer containing time
52BC 114140     LD      DE,4041H    Addr of time buffer in system area
52BF 010300     LD      BC,0003H    Number of bytes to move
52C2 EDB0       LDIR                Move time to system area
52C4 C9         RET                 Ret to SYS6 to read next command
*
*                       CLOCK Routine
*
52C5 CDA051     CALL    51A0H       SYS1 ON/OFF parameter processor
52C8 11A94C     LD      DE,4CA9H    Clock maintenance routine address
52CB 3E06       LD      A,06H       Index for clock routine in
                                    interrupt service list
52CD CA1344     JP      Z,4413H     Remove addr from list, go to SYS1
52D0 C31044     JP      4410H       Add addr to list. Go to SYS1
*
*
*
52D3 21DF52     LD      HL,52DFH    Address of BAD FORMAT message
52D6 CD6744     CALL    4467H       Display message
52D9 C33040     JP      4030H       Re-execute SYS1 (get next command)
*
*                       Local Storage Area
*
52DC            DEFB    0           3 byte buffer used to hold date
52DD            DEFB    0           and time before transferring
52DE            DEFB    0           them to the system area
*
*
*
52DF 42         DEFM                'BAD FORMAT'
52E9 0D         DEFB    0DH         MESSAGE TERMINATOR
52EA 11DE52     LD      DE,52DEH    Ending addr of local 3 byte buffer
52ED 0603       LD      B,03H       Number of fields to process
52EF D5         PUSH    DE          Save storage address
52F0 CDFF52     CALL    52FFH       Process next field. Must be 2 digits
52F3 D1         POP     DE          Restore buffer address
52F4 C0         RET     NZ          Exit if error in field.
                                    Non-numeric encountered
52F5 12         LD      (DE),A      Save a date field
52F6 1B         DEC     DE          Backup to next storage address
52F7 05         DEC     B           Count one field processed
52F8 C8         RET     Z           Exit if all fields processed
52F9 7E         LD      A,(HL)      Get field separator from command
52FA 23         INC     HL          Bump to 1st char of next field
52FB B9         CP      C           Compare separator to specified value
52FC 28F1       JR      Z,52EFH     Loop if legitimate separator
52FE C9         RET                 Ret to caller if illegal separator
*
*
*
52FF CD1653     CALL    5316H       Get next char from command string
5302 3010       JR      NC,5314H    Jump if alpha character (error)
5304 5F         LD      E,A         Save value (will be added later)
5305 07         RLCA                Value * 2
5306 07         RLCA                Value * 4
5307 83         ADD     A,E         Value * 5
5308 07         RLCA                Value * 10
5309 5F         LD      E,A         Save binary equivalent of digit
530A CD1653     CALL    5316H       Get next char from command string
530D 3005       JR      NC,5314H    Jump if not numeric (error)
530F 83         ADD     A,E         Combine with value of previous
5310 5F         LD      E,A         Digit * 10
5311 AF         XOR     A           Set status flag for no error
5312 7B         LD      A,E         Binary equivalent of decimal
                                    value in A-reg.
5313 C9         RET                 Return to caller
5314 B7         OR      A           Signal error with non-zero status
5315 C9         RET                 Ret. Alpha character encountered
*
*
*
5316 7E         LD      A,(HL)      Fetch real char from command buffer
5317 23         INC     HL          Bump to next character
5318 D630       SUB     30H         Convert ASCII digit to binary
531A FE0A       CP      0AH         Test for non-numeric character
531C C9         RET                 Ret w/CARRY if char greater than 9
*
*                       DEVICE Routine
*
531D 21C043     LD      HL,43C0H    Device table address
5320 7E         LD      A,(HL)      .<- Get ASCII char of device type
5321 2C         INC     L           .  | Bump to next char in device name
5322 B7         OR      A           .  | Check for end of list
5323 2812       JR      Z,5337H     .--| Jump if end of device table, else
5325 CD3300     CALL    0033H       .  | display 1st char of device type
5328 7E         LD      A,(HL)      .  | Get 2nd char of device type
5329 CD3300     CALL    0033H       .  | Display it then
532C 3E0D       LD      A,0DH       .  | follow with car. ret. to
532E CD3300     CALL    0033H       .  | skip to next line
5331 7D         LD      A,L         .  | LSB of device table address
5332 C603       ADD     A,03H       .  | Add 3 to get next entry.
                                    .  | Skip over driver address
5334 6F         LD      L,A         .  | Reform addr of next entry in HL
5335 30E9       JR      NC,5320H    .--| Test if absolute end of
                                    .  | list reached (43CF)
5337 C32D40     JP      402DH       .<- Return to SYS1
*
*                       LIB Routine
*
533A 21BD4E     LD      HL,4EBDH    Addr of 2nd cmd list in SYS1
533D 0E04       LD      C,04H       .   No. of cmds printed per line
533F 3EC2       LD      A,0C2H      .   Compression code for 2 blanks
5341 CD3300     CALL    0033H       .   Display two blanks
5344 0606       LD      B,06H       .   No. chars in each cmd list entry
5346 7E         LD      A,(HL)      .<- Get a char from command list
5347 23         INC     HL          .  | Bump to next char in list
5348 CD3300     CALL    0033H       .  | Display character
534B 10F9       DJNZ    5346H       .--| Loop till all chars displayed
534D 3EC8       LD      A,0C8H      .   Compression code for 8 blanks
534F CD3300     CALL    0033H       .   Display 8 blanks
5352 23         INC     HL          .   Skip over addr of action routine
5353 23         INC     HL          .   in SYS1 for this command
5354 7E         LD      A,(HL)      .   Get 1st char of next command
5355 B7         OR      A           .   Get status for end of list test
5356 280A       JR      Z,5362H     .   Jump if end of list reached
```

```
5358 0D         DEC     C               .   Count no. of cmds on this line
5359 20E9       JR      NZ,5344H        .   Jump if line not full
535B 3E0D       LD      A,0DH           .   Else, skip to next line
535D CD3300     CALL    0033H           .   By displaying carriage return
5360 18DB       JR      533DH           .   Go top of loop, reset counts
5362 3E0D       LD      A,0DH           Done, skip
5364 CD3300     CALL    0033H           to next line
5367 C32D40     JP      402DH           and ret to SYS1 for next command
*
*                       PROT Command Processing
*
536A 3A0F43     LD      A,(430FH)       Get system condition/permission flag
536D CB6F       BIT     05H,A           Test if PROT command allowed
536F C26E54     JP      NZ,546EH        Jump if PROT not permitted
5372 010000     LD      BC,0000H        Zero value for
5375 ED43D953   LD      (53D9H),BC      PW flag
5379 ED43F953   LD      (53F9H),BC      LOCK flag
537D ED430354   LD      (5403H),BC      UNLOCK flag
5381 0E00       LD      C,00H           Setup C reg for default drive 0
5383 7E         LD      A,(HL)          Get 1st char following PROT command
5384 FE3A       CP      3AH             If it's a : there's a drive spec.
5386 2006       JR      NZ,538EH        Jump if no drive specification
5388 23         INC     HL              Get drive no. Bump to next char
5389 7E         LD      A,(HL)          Fetch drive number from command
538A D630       SUB     30H             Convert ASCII to binary
538C 4F         LD      C,A             Save drive number in C reg
538D 23         INC     HL              Bump to next char in command
538E 79         LD      A,C             Drive number to A register
538F 32BC53     LD      (53BCH),A       so it can be saved at 53BC
5392 11EB54     LD      DE,54EBH        Address of PW/LOCK/UNLOCK options
5395 CD7644     CALL    4476H           SYS1/code 6, compare rest of
                                        command to option list
5398 3AD953     LD      A,(53D9H)       Get PW flag
539B 21F953     LD      HL,53F9H        Address of LOCK flag
539E B6         OR      (HL)            Combine PW/LOCK test results (SYS1)
539F 210354     LD      HL,5403H        Address of UNLOCK flag
53A2 B6         OR      (HL)            Combine with PW/LOCK/UNLOCK test
                                        results from SYS1
53A3 CA6B52     JP      Z,526BH         Go if PW/LOCK/UNLOCK not specified
53A6 3E03       LD      A,03H           Illegal character code
53A8 115155     LD      DE,5551H        Address of local DCB
53AB 12         LD      (DE),A          Put illegal char in DCB so OPEN
                                        will return immediately
53AC CD2444     CALL    4424H           OPEN dummy file. Load SYS2 into 4E00
53AF 21A054     LD      HL,54A0H        Addr of MASTER PASSWORD? message
53B2 CD6744     CALL    4467H           Display message
53B5 CD5454     CALL    5454H           Go read password from keyboard
53B8 226851     LD      (5168H),HL      Save encode of password
53BB 0E00       LD      C,00H           C = drive number to read
53BD CDF04A     CALL    4AF0H           Read track 11, sector 0 into 4D00
53C0 C24C55     JP      NZ,554CH        Jump if error during READ
53C3 2ACE4D     LD      HL,(4DCEH)      Addr of password in GAT sector buffer
53C6 ED5B6851   LD      DE,(5168H)      Addr of new PW in SYS2 storage area
53CA AF         XOR     A               Clear CARRY flag
53CB ED52       SBC     HL,DE           Compare encode of passwords
53CD 2809       JR      Z,53D8H         Go if correct PW entered, 53CF 21D254
LD      HL,54D2H     Addr of INVALID MASTER PASSWORD msg
53D2 CD6744     CALL    4467H           Display message
53D5 C33040     JP      4030H           Return to SYS1 for next command
*
*
*
53D8 110000     LD      DE,0000H        Load results from SYS1 PW test
53DB 7A         LD      A,D             53D9/53DA will contain
53DC B3         OR      E               FFFF if PW option detected
53DD 2816       JR      Z,53F5H         Go if zero (PW option not selected)
53DF 21B954     LD      HL,54B9H        Addr of NEW MASTER PASSWORD? msg
53E2 CD6744     CALL    4467H           Display message
53E5 CD5454     CALL    5454H           Put reply (PW) in buffer at 5600
53E8 22CE4D     LD      (4DCEH),HL      Save PW encode in GAT sector buffer
53EB 3ABC53     LD      A,(53BCH)       Get drive number
53EE 4F         LD      C,A             Move to C register for nucleus call
53EF CD034B     CALL    4B03H           Write GAT sector
53F2 C24C55     JP      NZ,554CH        Go if error during GAT sector write
53F5 2ACE4D     LD      HL,(4DCEH)      Load encode of password
53F8 110000     LD      DE,0000H        Get SYS flag for LOCK option
53FB 7A         LD      A,D             53F9-53FA will contain
53FC B3         OR      E               FFFF if LOCK specified
53FD 200D       JR      NZ,540CH        Jump if LOCK specified
53FF 219642     LD      HL,4296H        Encode of UNIVERSAL password
5402 110000     LD      DE,0000H        Get SYS1 flag for UNLOCK
5405 7A         LD      A,D             Combined SYS1
5406 B3         OR      E               flag bytes
5407 2003       JR      NZ,540CH        Go if UNLOCK specified, else
5409 C32D40     JP      402DH           Go SYS1 (TRSDOS READY) No PROT cmd
540C 223054     LD      (5430H),HL      Save LOCK/UNLOCK PW encode specified
540F 3ABC53     LD      A,(53BCH)       Load drive number
5412 4F         LD      C,A             Move to C for nucleus call
5413 CD554B     CALL    4B55H           Get dir track no. for drive in C
5416 1E02       LD      E,02H           Sector number to read
5418 210042     LD      HL,4200H        Buffer address = 4200
541B CD354B     CALL    4B35H           Read track 11, sector 2
541E C24C55     JP      NZ,554CH        Jump if error during read
5421 2E40       LD      L,40H           Form addr of 3rd entry in directory
5423 7E         LD      A,(HL)          Get entry available/assigned byte
5424 E6F8       AND     0F8H            Isolate available/assigned list
5426 FE10       CP      10H             Test if entry available
5428 2010       JR      NZ,543AH        Jump if entry assigned
542A 7D         LD      A,L             Get LSB of beginning of entry addr
542B C610       ADD     A,10H           Compute addr of UPDATE password
542D 6F         LD      L,A             Reform address in HL
542E D5         PUSH    DE              Save track/sector number
542F 110000     LD      DE,0000H        Load master password
5432 73         LD      (HL),E          Move it to UPDATE password (LSB)
5433 2C         INC     L               Bump to next password in entry
5434 72         LD      (HL),D          MSB of master pw to MSB of UPDATE pw
5435 2C         INC     L               Bump to LSB of access pw
5436 73         LD      (HL),E          LSB of master pw to LSB of access pw
5437 2C         INC     L               Bump to MSB of PW
5438 72         LD      (HL),D          MSB of master PW to MSB of ACCESS PW
5439 D1         POP     DE              Restore track/sector no. for next read
543A 7D         LD      A,L             Prepare to form addr of next entry
543B E6E0       AND     0E0H            Test if end of sector reached
543D C620       ADD     A,20H           Compute address of next entry
543F 6F         LD      L,A             Form address in HL
5440 30E1       JR      NC,5423H        Go if end of buffer not reached
5442 210042     LD      HL,4200H        Reform buffer address
5445 CDEF46     CALL    46EFH           Write updated sector back to disk
5448 C24C55     JP      NZ,554CH        Jump if error during rewrite
544B 1C         INC     E               Bump sector address
544C 7B         LD      A,E             and move it to A where end
544D FE0A       CP      0AH             of track test can be made
544F 38C7       JR      C,5418H         Jump if not end of track
5451 C32D40     JP      402DH           Done, ret to SYS1 for next command
5454 0608       LD      B,08H           Max no. of characters to read
5456 210056     LD      HL,5600H        Address of local buffer
5459 CD4000     CALL    0040H           Wait for user input
545C EB         EX      DE,HL           DE = beginning buffer address
545D 2600       LD      H,00H           Set HL equal to number of
545F 68         LD      L,B             characters read from keyboard
5460 19         ADD     HL,DE           Compute end of data in buffer
5461 3E08       LD      A,08H           Then compute number of bytes
5463 90         SUB     B               remaining in buffer
5464 47         LD      B,A             and save as counter
5465 3E20       LD      A,20H           Then blank fill
5467 77         LD      (HL),A          rest of buffer
5468 23         INC     HL              Bump to next position in buffer
5469 10FC       DJNZ    5467H           and loop till buffer blank
546B C3D150     JP      50D1H           SYS2 subroutine to encode password
*
*
*
546E 217754     LD      HL,5477H        Address of INVALID COMMAND
                                        DURING PROGRAM CHAINING msg.
5471 CD6744     CALL    4467H           Display message
5474 C33040     JP      4030H           and return to SYS1 for next command
*
*
*
5477            DEFB    0AH             Line feed
5478            DEFM    'INVALID COMMAND DURING PROGRAM CHAINING'
549F            DEFB    0DH             Message terminator
54A0            DEFB    0AH             Line feed
54A1            DEFM    'MASTER PASSWORD ?      '
54B8            DEFB    03H             Message terminator
54B9            DEFB    0AH             Line feed
54BA            DEFM    'NEW MASTER PASSWORD ?   '
54D1            DEFB    03H             Message terminator
54D2            DEFM    'INVALID MASTER PASSWORD'
54EA            DEFB    0DH             Message terminator
*
*               Text Strings for PROT Parameter
*
54EB            DEFM    'PW    '        Parameter
54F1            DEFW    53D9H           Address for password value
54F3            DEFM    'LOCK  '        Parameter
54F9            DEFW    53F9H           Address for lock parameter
54FB            DEFM    'UNLOCK'        Parameter
5501            DEFW    5403H           Address for unlock parameter
5503            DEFB    00H             Text string terminator
*
*             Verify Command Processing
*
5504 CDA051     CALL    51A0H           SYS1 routine to check for ON/OFF
5507 21B847     LD      HL,47B8H        Turn verify flag ON or OFF
550A 2803       JR      Z,550FH         Jump if VERIFY (OFF)
550C 21A847     LD      HL,47A8H        Move addr of VRFY routine in nucleus
550F 223A44     LD      (443AH),HL      to addr field of WRITE jump vector
5512 C32D40     JP      402DH            Done. Ret to SYS1 for next command
*
*  Pause routine. Reads keyboard and tests for SHIFT @
*  key active. If active, pauses until any key struck.
*
*  Called by APPEND, LIST, PRINT and DIR.
*
*  If Bit 5 in the system condition flag (430F) is not set,
*  return to the caller with the zero status flag set.
*  if Bit 5 is set then:
*
*         1. Test the BREAK key. If active return with a
*            non-zero status.
*         2. If the BREAK key is inactive, strobe the
*            keyboard.
*            If the SHIFT @ keys are inactive, return with a
*            zero status. If the SHIFT @ keys are active,
*            loop till any key becomes active and return
*            control to the caller with the following status:
*
*                 BREAK active - non-zero status
*                 Any key active - zero status
*
*
5515 3A0F43     LD      A,(430FH)       Get system condition flags
5518 E620       AND     20H             Isolate BREAK key test flag
551A EE20       XOR     20H             Test if BREAK test flag on
551C C8         RET     Z               Ret to caller if BREAK test inactive
551D 3A4038     LD      A,(3840H)       Load row contents for BREAK key
5520 E604       AND     04H             Isolate BREAK key list
5522 C0         RET     NZ              Exit non-zero status if BREAK active
5523 CD2B00     CALL    002BH           BREAK not active. Strobe keyboard
5526 FE60       CP      60H             Test for SHIFT @
5528 2802       JR      Z,552CH         Jump if SHIFT @ active
552A AF         XOR     A               Else signal no activity
552B C9         RET                     and return to caller
552C 3A4038     LD      A,(3840H)       Refetch BREAK row contents
552F E604       AND     04H             Isolate BREAK key bit
5531 C0         RET     NZ              Exit if BREAK active
5532 CD2B00     CALL    002BH           If not active strobe keyboard again
5535 B7         OR      A               Set status flag for input
```

```
5536 28F4      JR      Z,552CH      Loop till BREAK active (return
                                    w/non-zero status)
5538 AF        XOR     A            Or, any key active (return
                                    w/zero status)
5539 C9        RET                  Return to caller
*
*
*
553A 219155    LD      HL,5591H     Address of FILE SPEC REQUIRED message
553D CD6744    CALL    4467H        Display message
5540 C33040    JP      4030H        Recall SYS1 and wait for next command
*
*
*
554C F640      OR      40H          Add file error bit to error code
                                    in A-register.
554E C30944    JP      4409H        Call SYS4 to print error message
*
*
*
5551           DEFS    20H          Local DCB
*
*
*
5591           DEFB    0AH          Line feed
5592           DEFM    'FILE SPEC REQUIRED
55A4           DEFB    0DH          Message terminator
55A5           DEFB    0AH          Line feed
55A6           DEFM    'DEVICE SPEC REQUIRED'
55BA           DEFB    0DH          Message terminator
*
*          APPEND command processing
*
5700 115155    LD      DE,5551H     Address of local DCB area
5703 CD1C44    CALL    441CH        Copy/edit file into DCB (SYS1, code 4)
5706 C23A55    JP      NZ,553AH     Display FILE SPEC. REQUIRED
                                    (error if no file name)
5709 1A        LD      A,(DE)       Fetch 1st character of file name
570A FE2A      CP      2AH          Test for reference to special file
570C CA3A55    JP      Z,553AH      Display FILE SPEC REQUIRED
                                    (special file reference)
570F 117155    LD      DE,5571H     Address of 2nd local DCB area
5712 CD1C44    CALL    441CH        Copy 2nd file name to DCB area
5715 C23A55    JP      NZ,553AH     Display FILE SPEC REQUIRED
                                    (error in 2nd file name)
5718 1A        LD      A,(DE)       Get first char of 2nd file name
5719 FE2A      CP      2AH          Test for reference to special file
571B CA3A55    JP      Z,553AH      Display FILE SPEC REQUIRED
                                    (special file reference)
571E 0600      LD      B,00H        Specify physical I/O
5720 210056    LD      HL,5600H     Sector buffer addr for 2nd file
5723 CD2444    CALL    4424H        OPEN 2nd file
5726 C24C55    JP      NZ,554CH     Jump if any error during OPEN
5729 0600      LD      B,00H        Specify physical I/O
572B 115155    LD      DE,5551H     DCB address of 1st file
572E 21006F    LD      HL,6F00H     Sector buffer addr for first file
5731 CD2444    CALL    4424H        OPEN second file
5734 C24C55    JP      NZ,554CH     Go if error while OPENing 2nd file
5737 117155    LD      DE,5571H     DCB address for 2nd file
573A CD4844    CALL    4448H        Skip to end of file on 2nd file
573D 00        NOP
573E 00        NOP
573F 00        NOP
5740 00        NOP
5741 00        NOP
5742 00        NOP
5743 00        NOP
5744 00        NOP
5745 00        NOP
5746 00        NOP
5747 00        NOP
5748 00        NOP
5749 00        NOP
574A 00        NOP
574B 00        NOP
574C C3B657    JP      57B6H        Copy file 1 onto end of file 2
*
*          COPY Command Processing
*
574F 115155    LD      DE,5551H     Address of local DCB area
5752 CD1C44    CALL    441CH        SYS1, code 4, copy/edit 1st file
5755 C23A55    JP      NZ,553AH     Go if illegal chars in file name
5758 117155    LD      DE,5571H     Address of 2nd local DCB area
575B CD1C44    CALL    441CH        SYS1, code 4, copy/edit 2nd file name
575E C23A55    JP      NZ,553AH     Jump if illegal chars 2nd file name
5761 0600      LD      B,00H        Set physical I/O for source file
5763 115155    LD      DE,5551H     Address of source DCB
5766 210056    LD      HL,5600H     Sector buffer address
5769 CD2444    CALL    4424H        OPEN source file
576C C24C55    JP      NZ,554CH     Jump if any error during OPEN
576F 0600      LD      B,00H        Set physical I/O for destination file
5771 117155    LD      DE,5571H     Destination DCB address
5774 210056    LD      HL,5600H     Share sector buf with source file
5777 CD2044    CALL    4420H        INIT call for destination file
577A C24C55    JP      NZ,554CH     Jump if error during INIT processing
577D 115155    LD      DE,5551H     .  Address of source DCB
5780 CD3644    CALL    4436H        .  Read sector from source file
5783 280B      JR      Z,5790H      .  Jump if no error during read
5785 FE1C      CP      1CH          .  Test for EOF on source file
5787 2818      JR      Z,57A1H      .  Jump if EOF on source
5789 FE1D      CP      1DH          .  else test for record not found
578B 280E      JR      Z,579BH      .  Jump if record not found
578D C34C55    JP      554CH        .  Some other error. Call SYS4
                                    .  to display error message
5790 117155    LD      DE,5571H     .  No error on source file.
                                    .  Load address of dest. DCB
5793 CD3944    CALL    4439H        .  Write sector just read to
                                    .  destination file
5796 28E5      JR      Z,577DH      .  Go if no error during WRITE
5798 C34C55    JP      554CH        Loop till EOF on source file
579B 2A5955    LD      HL,(5559H)   Record not found. Get record no.
579E 227955    LD      (5579H),HL   from source DCB, copy to dest. DCB
57A1 117155    LD      DE,5571H     Address of destination DCB
57A4 CD2844    CALL    4428H        CLOSE destination file
57A7 C24C55    JP      NZ,554CH     Jump if any error during CLOSE
57AA 115155    LD      DE,5551H     Address of source DCB
57AD CD2844    CALL    4428H        CLOSE source file
57B0 C24C55    JP      NZ,554CH     Jump if error during CLOSE
57B3 C32D40    JP      402DH        Return to SYS1 wait for next command
*
*          Continuation of APPEND Processing
*
57B6 115155    LD      DE,5551H     DCB address for first file
57B9 CD1300    CALL    0013H        Read a character from 1st file
57BC 2807      JR      Z,57C5H      Jump if no error condition
57BE FE1C      CP      1CH          An error occurred. Was it end of file
57C0 28DF      JR      Z,57A1H      Jump if yes (EOF or source file
57C2 C34C55    JP      554CH        If not, Call SYS4 to process error
57C5 47        LD      B,A          Preserve character just read
57C6 CD1555    CALL    5515H        Test for BREAK/SHIFT@
57C9 20D6      JR      NZ,57A1H     If BREAK key active terminate operation
57CB 78        LD      A,B          Restore character read
57CC 117155    LD      DE,5571H     DCB addr of 2nd file (dest. file)
57CF CD1B00    CALL    001BH        Write char from source to dest. file
57D2 C24C55    JP      NZ,554CH     Jump if error during write operation
57D5 18DF      JR      57B6H        Else loop till EOF on source file
*
*          DUMP Command Processing
*
57D7 010070    LD      BC,7000H     Default START address DUMP
57DA ED436858  LD      (5868H),BC   Initialize START address location
57DE ED436E58  LD      (586EH),BC   Initialize END address location
57E2 0120D40   LD      BC,402DH     Default TRANSFER address
57E5 ED43AB58  LD      (58ABH),BC   Initialize TRANSFER addr location
57E9 115155    LD      DE,5551H     Address of local DCB area
57EC CD1C44    CALL    441CH        SYS1, code 4, copy/edit file name
57EF C23A55    JP      NZ,553AH     Jump if illegal chars in name
57F2 1A        LD      A,(DE)       None, check for
57F3 FE2A      CP      2AH          reference to special file (*)
57F5 CA3A55    JP      Z,553AH      Jump if found (error)
57F8 11F558    LD      DE,58F5H     Address of START text
57FB CD7644    CALL    4476H        SYS1, code 6. Look for word START
57FE 110E59    LD      DE,590EH     Address of local buffer area
5801 215155    LD      HL,5551H     Address of local DCB
*
*     Copy 1st 6 characters of file name to 590E. Copy
*     terminates when a special character is detected or
*     6 characters have been copied
*
5804 0606      LD      B,06H        Max no. of chars to copy
5806 7E        LD      A,(HL)       .  Get a char from file name
5807 FE30      CP      30H          .  Test for special character
                                    .  (end of file name)
5809 3813      JR      C,581EH      .  Go if end of file name found
580B FE3A      CP      3AH          .  Test for numeric 0-9
580D 3808      JR      C,5817H      .  Jump if 0-9. Copy character
580F FE41      CP      41H          .  Test for special char 3A-3F
5811 380B      JR      C,581EH      .  Go if drive no. found. End copy
5813 FE5B      CP      5BH          .  Test for upper case letters
5815 3007      JR      NC,581EH     .  Go if upper case found. End copy
5817 12        LD      (DE),A       .  Char in A-Z, 0-9 range. Copy to
5818 23        INC     HL           .  Short list (590E) then bump
5819 13        INC     DE           .  Fetch and store address
581A 10EA      DJNZ    5806H        .  Loop till 6 chars copied
                                    .  or end of name found
581C 1806      JR      5824H        .  6 chars copied. Skip blank file
581E 3E20      LD      A,20H        .  ASCII blank to fill short list
5820 12        LD      (DE),A       .  Add to short list
5821 13        INC     DE           .  Bump to next pos. in short list
5822 10FA      DJNZ    581EH        .  Loop till 6 char name blanked
5824 115155    LD      DE,5551H     .  Addr of DCB for DUMP file
5827 21F258    LD      HL,58F2H     Address CIM text
582A CD7344    CALL    4473H        SYS1/code 5. Add CIM extension
582D 0600      LD      B,00H        Specify physical I/O
582F 210056    LD      HL,56        sector buffer address
5832 CD2044    CALL    4420H        INIT file
5835 C24C55    JP      NZ,554CH     Jump if error during INIT call
5838 3E05      LD      A,05H        Control code for file name
583A CD1B00    CALL    001BH        Send control code to disk file
583D 3E06      LD      A,06H        Number of chars in file name
583F CD1B00    CALL    001BH        Send byte count to disk file
5842 0606      LD      B,06H        Count for writing file name
5844 210E59    LD      HL,590EH     Address of file name buffer
5847 7E        LD      A,(HL)       Fetch a character from file name
5848 23        INC     HL           Bump to next character in name
5849 CD1B00    CALL    001BH        Send file name char to disk file
584C 10F9      DJNZ    5847H        Loop till name written (6 chars)
584E 2A6E58    LD      HL,(586EH)   Load END addr given in DUMP command
5851 ED4B6858  LD      BC,(5868H)   Load START addr given in DUMP command
5855 AF        XOR     A            Clear carry
5856 ED42      SBC     HL,BC        END addr minus start address
5858 21C458    LD      HL,58C4H     Addr END LESS THAN START message
585B 3861      JR      C,58BEH      Jump if END less than START (error)
585D 21FF6F    LD      HL,6FFFH     Check for START address over 7000
5860 ED42      SBC     HL,BC        Compute 7000 minus START address
5862 21D958    LD      HL,58D9H     Addr START LESS THAN 7000 error msg.
5865 3057      JR      NC,58BEH     Go if START less than 7000 (error)
5867 210000    LD      HL,0000H     START address. Filled in by SYS1
                                    from call at 57FB
586A E5        PUSH    HL           .  Save current START address
586B 44        LD      B,H          .  Then copy it
586C 4D        LD      C,L          .  To BC
586D 210000    LD      HL,0000H     .  END address. Filled in by
                                    .  SYS1 from call at 57FB
5870 23        INC     HL           .  END addr +1 for true byte count
5871 AF        XOR     A            .  Clear CARRY
5872 ED42      SBC     HL,BC        .  Compute no. of bytes to write
5874 2829      JR      Z,589FH      .  Go if done. Else break transfer
                                    .  into blocks of 254 bytes
5876 06FE      LD      B,0FEH       .  Max bytes between control bytes
```

```
5878 7C          LD      A,H             .   Test byte count for transfer
5879 B7          OR      A               .   Get status for upper 8 bits
587A 2006        JR      NZ,5882H        .   Go if transfer count > 255
587C 7D          LD      A,L             .   Transfer not > 255
587D FEFF        CP      0FFH            .   Check if = 255
587F 3001        JR      NC,5882H        .   Jump if it is. Use 254
5881 45          LD      B,L             .   Move count < 255. Use true val
5882 E1          POP     HL              .   Restore begin. addr current block
5883 3E01        LD      A,01H           .   Code 01 (load data follows)
5885 CD1B00      CALL    001Bh           .   Write control code 01
5888 78          LD      A,B             .   Get count. Add 2 for overhead
5889 C602        ADD     A,02H           .   If it was 254 (max), may write
588B CD1B00      CALL    001BH           .   a zero. Write count for DATA
588E 7D          LD      A,L             .   Then write LSB
588F CD1B00      CALL    001BH           .   of load address
5892 7C          LD      A,H             .   followed by MSB
5893 CD1B00      CALL    001BH           .   of load address
5896 7E          LD      A,(HL)          .   Then begin writing core image
5897 23          INC     HL              .   Bump to next fetch address
5898 CD1B00      CALL    001BH           .   Write a byte of core image
589B 10F9        DJNZ    5896H           .   Loop till block written
589D 18CB        JR      586AH           .   Go test if all blocks written
589F E1          POP     HL              .   Done, clear stack, prepare to
58A0 3E02        LD      A,02H           write transfer address. First
58A2 CD1B00      CALl    001BH           Write a control code 02
58A5 3E02        LD      A,02H           followed by a byte count
58A7 CD1B00      CALL    001BH           for the address (2 of course)
58AA 210000      lD      HL,0000H        Transfer address filled in by
                                         SYS1 from call at 57FB
58AD 7D          Ld      A,L             Fetch a LSB of transfer address
58AE CD1B00      CALL    001BH           Write LSB of transfer address
58B1 7C          LD      A,H             Fetch MSB of transfer address
58B2 CD1B00      CALL    001BH           Write MSB of transfer address
58B5 CD2844      CALL    4428H           CLOSE disk file.
58B8 C24C55      JP      NZ,554CH        Jump if any error during CLOSE
58BB C32D40      JP      402DH           No error jump to SYS1
58BE CD6744      CALL    4467H           Write DUMP error message
58C1 C33040      JP      4030H           Ret to SYS1. Wait for next command
*
*                DUMP Error Messages
*
58C4             DEFB    0AH             Line feed
58C5             DEFM                    'END LESS THAN START'
58D8             DEFB    0DH             Message terminator
58D9             DEFB    0AH             Line feed
58DA             DEFM                    'START LESS THAN X'7000''
58F1             DEFB    0DH             Message terminator
*
*
58F2             DEFM    'CIM'           Extension for DUMP file name
*
*                DUMP Parameter Text Strings
*
58F5             DEFM    'START '
58FA             DEFW    58FAH           Location for START address
58FD             DEFM    'END   '
5902             DEFW    586EH           Location for END addresss
5905             DEFM    'TRA   '
590A             DEFW    58ABH           Location for TRA address
590D             DEFB    00              List terminator
*
*
*
590E             DEFS    6               Temporary buffer area
*
*                KILL Command Processing
*
5914 115155      LD      DE,5551H        Local DCB address
5917 CD1C44      CALL    441CH           File name to local DCB (SYS1 routine)
591A C23A55      JP      nZ,553AH        Jump if illegal chars in file name
591D CD2444      CALL    4424H           OPEN file to KILL
5920 C24C55      JP      NZ,554CH        Jump if any error during OPEN
5923 CD2C44      CALL    442CH           KILL file
5926 C24C55      JP      NZ,554CH        Jump if error during KILL
5929 C32D40      JP      402DH           Reload SYS1 to wait for next command
*
*                LIST Command Processing
*
592C 010000      LD      BC,0000H        Signal no LINE option
592F ED435459    LD      (5954H),BC      Selected by zeroing 5954
5933 115155      LD      DE,5551H        Address of local DCB
5936 CD1C44      CALL    441CH           Copy and edit file. Move it to
                                         local DCB. Use SYS1 subroutine.
5939 C23A55      JP      NZ,553AH        Jump if illegal chars in file name
593C 1A          LD      A,(DE)          Test for
593D FE2A        CP      2AH             Reference to special file (*)
593F CA3A55      JP      Z,553AH         Go if special file reference (error)
5942 018B59      LD      BC,598BH        Address of LINE text
5945 CD7644      CALL    4476H           SYS1, option 6 test for LINE option
5948 0600        LD      B,00H           Specify physical I/O
594A 215056      LD      HL,5600H        Sector buffer address for file
594D CD2444      CALL    4424H           OPEN file to be LIST'ed
5950 C24C55      JP      NZ,554CH        JUMP if error during OPEN
5953 010000      LD      BC,0000H        LINE option flag
5956 78          LD      A,B             BC = 0000 if LINE not specified
5957 B1          OR      C               BC = Value if LINE value specified
5958 280D        JR      Z,5967H         Jump if LINE option not given
595A CD1300      CALL    0013H           Get a character from source file
595D FE0D        CP      0DH             Have we reached end of a line
595F 20F9        JR      NZ,595AH        No, loop till end of line
5961 0D          DEC     C               Yes, decrement line count
5962 20F6        JR      NZ,595AH        in BC. If non-zero
5964 05          DEC     B               Keep looping (skipping lines)
5965 20F3        JR      NZ,595AH        till no. in LINE option passed
5967 115155      LD      DE,5551H        Addr of file DCB. Begin listing file
596A CD1300      CALL    0013H           Get a character from source file
596D 200F        JR      NZ,597EH        Go if not good status (go test EOF)
596F CD3300      CALL    0033H           Display character returned
5972 FE0D        CP      0DH             Test for end of line
5974 20F1        JR      NZ,5967H        If not end, loop till end found
5976 CD1555      CALL    5515H           Test for BREAK/SHIFT@ condition
5979 C23040      JP      NZ,4030H        If BREAK active terminate LIST
597C 18E9        JR      5967H           Else, loop till end of source file
597E FE1C        CP      1CH             Test error status for EOF
5980 CA2D40      JP      Z,402DH         If so, Get SYS1 for next command
5983 FE1D        CP      1DH             Test for record not found error
5985 CA2D40      JP      Z,402DH         If RNF, done. Get SYS1
5988 C34C55      JP      554CH           Call SYS4 to generate error msg.
*
*                LIST Parameter Text List
*
598B             DEFM    'LINE  '
5991 54          DEFW    5991H           Address of LIST command routine
5993             DEFB    00H             List terminator
*
*                   LOAD command processor
*
5994 115155      LD      DE,5551H        Address of local DCB area
5997 CD1C44      CALL    441CH           SYS1/code 4, copy file name to DCB
599A C23A55      JP      NZ,553AH        Jump if illegal chars found in name
599D 1A          LD      A,(DE)          else test for special file
599E FE2A        CP      2AH             name (*)
59A0 CA3A55      JP      Z,553AH         Go if special file name * (error)
59A3 CD3044      CALL    4430H           Call nucleus routine to load file
                                         (hope it doesn't overlay SYS6)
59A6 CA2D40      JP      Z,402DH         No error during load. Recall SYS1
59A9 C34C55      JP      554CH           Add file bit to error code and
                                         call SYS 4. Wait for next command
*
*                PRINT Command Processor
*
59AC 115155      LD      DE,5551H        Address of local DCB area
59AF CD1C44      CALL    441CH           SYS1/code 4, copy file name to DCB
59B2 C23A55      JP      NZ,553AH        Go if illegal chars in name (error)
59B5 1A          LD      A,(DE)          Fetch 1st character of file name
59B6 FE2A        CP      2AH             Test for special file name (*)
59B8 CA3A55      JP      Z,553AH         Jump if special file name (error)
59BB 0600        LD      B,00H           Specify physical I/O
59BD 215056      LD      HL,5600H        Sector buffer address
59C0 CD2444      CALL    4424H           OPEN file
59C3 C24C55      JP      NZ,554CH        Jump if error occurred during OPEN
59C6 115155      LD      DE,5551H        Address of disk file DCB
59C9 CD1300      CALL    0013H           Get a character from source file
59CC 200C        JR      NZ,59DAH        If error while reading source file
                                         go test for EOF/RECORD NOT FOUND
59CE CD3B00      CALL    003BH           Send char to ROM print driver
59D1 CD1555      CALL    5515H           Test for BREAK or SHIFT @ condition
59D4 C23040      JP      NZ,4030H        If BREAK active exit print operation
                                         If SHIFT @ active wait for key hit
59D7 C3C659      JP      59C6H           Loop till disk read status non-zero
59DA F5          PUSH    AF              Save disk status
59DB 3E0D        LD      A,0DH           Terminate current line by
59DD CD3B00      CALL    003BH           Printing a carriage return
59E0 F1          POP     AF              Reload disk drive status
59E1 FE1C        CP      1CH             EOF on source file?
59E3 CA2D40      JP      Z,402DH         If so, ret to SYS1 (TRSDOS READY)
59E6 FE1D        CP      1DH             If not, check for RECORD NOT FOUND
59E8 CA2D40      JP      Z,402DH         If RNF ret to SYS1 for next command
59EB C34C55      JP      554CH           Not EOF/RNF, let SYS4 process it
*
*                Attribute Command Processing
*
5B00 115155      LD      DE,5551H        Local DCB address
5B03 CD1C44      CALL    441CH           SYS1, code 4. File name to DCB
5B06 C23A55      JP      NZ,553AH        Jump if error detected in file name
5B09 1A          LD      A,(DE)          Is first character
5B0A FE2A        CP      2AH             of file an *
5B0C CA3A55      JP      Z,553AH         If so, display FILE SPEC. REQUIRED
5B0F E5          PUSH    HL              Save addr of parameters
5B10 0600        LD      B,00H           Specify physical I/0
5B12 21004D      LD      HL,4D00H        Sector buffer address
5B15 CD2444      CALL    4424H           OPEN file (load SYS2 at 4E00)
5B18 C24C55      JP      NZ,554CH        Jump if file does not exist
5B1B E1          POP     HL              Restore command string address
5B1C 3A5255      LD      A,(5552H)       Fetch permission flags for file
5B1F E607        AND     07H             Isolate access flags
5B21 3E37        LD      A,37H           Signal restricted access
                                         Invisible file, occupied entry
5B23 C24C55      JP      NZ,554CH        Jump if any permission required
5B26 AF          XOR     A               Load a zero and
5B27 32055C      LD      (5C05H),A       clear values to be combined with
5B2A 320D5C      LD      (5C0DH),A       the access and invisibility attributes
5B2D 7E          LD      A,(HL)          Get next char from command string
5B2E FE28        CP      28H             Test for (
5B30 2808        JR      Z,5B3AH         Jump if (. Start of parameters
5B32 FE20        CP      20H             Not (. Test for blank
5B34 C26B52      JP      NZ,526BH        Not ( or , so ignore command
5B37 23          INC     HL              Blank. Skip to next character
5B38 18F3        JR      5B2DH           Loop till ( or end of command found
5B3A 23          INC     HL              Bump past (
5B3B 7E          LD      A,(HL)          Get next char of parameter field
5B3C FE49        CP      49H             Test for I
5B3E 2857        JR      Z,5B97H         Jump if INVISIBLE specified
5B40 FE41        CP      41H             Not I, test for ACC
5B42 2863        JR      Z,5BA7H         Jump if A found. Go test for CC
5B44 FE55        CP      55H             Not I or A. Test for UPD
5B46 2874        JR      Z,5BBCH         Jump if U. Go test for PD
5B48 FE50        CP      50H             Not I, A, or U. Test for PROT
5B4A C2395C      JP      NZ,5C39H        None of above. Go display
                                         ATTRIBUTE error message
5B4D CDD15B      CALL    5BD1H           Parse PROT command  parameter
5B50 CA395C      JP      Z,5C39H         Error if not (= KILL/RENAME/WRITE
                                         WRITE/READ or PROT
5B53 E5          PUSH    HL              Save command string address
5B54 0607        LD      B,07H           Number of entries to compare
5B56 ED5B5551    LD      DE,(5155H)      Get 1st 2 chars of PROT parameter
5B5A 21615C      LD      HL,5C61H        List of possible parameters
5B5D 7E          LD      A,(HL)          .   Get one character from list
5B5E 23          INC     HL              .   Bump to next char in list
5B5F BB          CP      E               .   Compare char of parameter to list
```

```
5B60 CC695B     CALL    Z,5B69H         .   If equal, try matching others
5B63 23         INC     HL              .   else bump to next option in list
5B64 10F7       DJNZ    5B5DH           Loop till all options compared
5B66 C3395C     JP      5C39H           Error, no match w/PROT options
*
*               Test PROT parameters
*
5B69 7E         LD      A,(HL)          Fetch 2nd char from option test
5B6A BA         CP      D               Compare with 2nd char of PROT option
5B6B C0         RET     NZ              Exit if not equal. Try next option
5B6C F1         POP     AF              Clear ret addr from stack. Will
                                        ret to 5BF7/5B3A or 5C39
5B6D 78         LD      A,B             No. of options remaining in list
5B6E 3D         DEC     A               Decrement for matching option
5B6F FE05       CP      05H             Are we on RE (READ or RENAME)
5B71 200B       JR      NZ,5B7EH        Jump if not RE
5B73 3A5751     LD      A,(5157H)       Fetch 3rd char of PROT parameter.
                                        Test for RENAME
5B76 FE4E       CP      4EH             Test for N (RENAME not READ
5B78 3E05       LD      A,05H           Access flags for READ
5B7A 2002       JR      NZ,5B7EH        Jump if READ option
5B7C 3E02       LD      A,02H           Access flags for RENAME
5B7E 320B5C     LD      (5C0BH),A       Save access permission for PROT
5B81 E1         POP     HL              Restore command string address
5B82 0601       LD      B,01H           KILL/RENAME/READ/WRITE/EXECUTE access
5B84 3A055C     LD      A,(5C05H)       Byte 0 of dir entry w/access field 0
5B87 B0         OR      B               Merge with new access permission
5B88 32055C     LD      (5C05H),A       Save updated byte to update dir
5B8B 7E         LD      A,(HL)          Get next char from command string
5B8C FE29       CP      29H             Test for )
5B8E 2867       JR      Z,5BF7H         Jump if ) found (end of command)
5B90 FE2C       CP      2CH             No left paren. Test for comma
5B92 28A6       JR      Z,5B3AH         Go if , found (more parameters follow)
5B94 C3395C     JP      5C39H           else error in attribute spec.
*
*                   ATTRIB - I Processing
*
5B97 CDD15B     CALL    5BD1H           Parse rest of command
5B9A C2395C     JP      NZ,5C39H        Jump if command string contains
5B9D 3A0D5C     LD      A,(5C0DH)       more than ). Get access permission
5BA0 F608       OR      08H             from dir entry. Set invisible flag
5BA2 320D5C     LD      (5C0DH),A       Save updated access permission flags
5BA5 18DD       JR      5B84H           Go test for closing ) and write
                                        updated directory entry
*
*                 ATTRIB - ACC Processing
*
5BA7 CDD15B     CALL    5BD1H           Parse rest of command
5BAA CA395C     JP      Z,5C39H         Error if nothing follows A
5BAD E5         PUSH    HL              Save current cmd string address
5BAE 115551     LD      DE,5155H        Address of ACCESS password
5BB1 CDD150     CALL    50D1H           Get encode of ACCESS password
5BB4 226A51     LD      (516AH),HL      Save encode
5BB7 E1         POP     HL              Restore command string address
5BB8 0602       LD      B,02H           RENAME/WRITE/READ/EXECUTE access
5BBA 18C8       JR      5B84H           Go update access permission
                                        field in directory
*
*                 ATTRIB - UPD Processing
*
5BBC CDD15B     CALL    5BD1H           Parse rest of command
5BBF CA395C     JP      Z,5C39H         Jump if nothing follows U
5BC2 E5         PUSH    HL              Save command string address
5BC3 115551     LD      DE,5155H        Address of buffer UPDATE password
5BC6 CDD150     CALL    50D1H           Get encode of UPDATE password
                                        (use subroutine in SYS2)
5BC9 226851     LD      (5168H),HL      Save encode
5BCC E1         POP     HL              Restore command string address
5BCD 0604       LD      B,04H           Specify WRITE/READ/EXECUTE access
5BCF 18B3       JR      5B84H           Go update access permission
                                        in directory sector
*
*                   Parse PROT Parameters
*
5BD1 23         INC     HL              .<-- Bump to next addr in command
5BD2 7E         LD      A,(HL)          .   Fetch next character
5BD3 FE0D       CP      0DH             .   Test for end of line
5BD5 C8         RET     Z               .   Exit if end of line
5BD6 FE29       CP      29H             .   Test for closing paren
5BD8 C8         RET     Z               .   Exit if )
5BD9 FE2C       CP      2CH             .   Test for comma
5BDB C8         RET     Z               .---Exit if comma
5BDC FE3D       CP      3DH             Test for =
5BDE 20F1       JR      NZ,5BD1H        Not = , loop
5BE0 23         INC     HL              Bump to 1st char following =
5BE1 115551     LD      DE,5155H        Address of buffer area in SYS2
5BE4 0608       LD      B,08H           No. of buffer bytes to initialize
5BE6 D5         PUSH    DE              Save beginning buffer address
5BE7 C5         PUSH    BC              Save max no. of bytes in buffer
5BE8 3E20       LD      A,20H           Initialization value
5BEA 12         LD      (DE),A          .<-- Initialize a buffer byte
5BEB 13         INC     DE              .   Bump to next buffer byte
5BEC 10FC       DJNZ    5BEAH           .---Loop till buffer initialized
5BEE C1         POP     BC              Restore buffer size value in B
5BEF D1         POP     DE              Restore beginning buffer address
5BF0 CD8150     CALL    5081H           Copy password to buffer address
                                        in DE. Use subroutine in SYS2
5BF3 2B         DEC     HL              Back-up cmd $ pointer to terminator
5BF4 F601       OR      01H             Signal password found
5BF6 C9         RET                     and return to caller
5BF7 ED4B5755   LD      BC,(5557H)      Get dir sector pointer from DCB
5BFB CDC14A     CALL    4AC1H           Read dir sector into buffer
5BFE C24C55     JP      NZ,554CH        Jump if error during read
5C01 7E         LD      A,(HL)          Fetch 1st byte of dir entry
5C02 E6F8       AND     0F8H            Clear access permissions
5C04 1600       LD      D,00H           Load new access permission
5C06 CB42       BIT     00H,D           Test if all permission granted
5C08 2802       JR      Z,5C0CH         Jump if not
5C0A F600       OR      00H             Combine access permission bits
5C0C F600       OR      00H             With file visability attribute
5C0E 77         LD      (HL),A          Save updated access in dir entry
5C0F 7D         LD      A,L             Skip to UPDATE access password
5C10 C610       ADD     A,10H           by adding 10 to current dir address
5C12 6F         LD      L,A             for address of UPDATE password
5C13 CB52       BIT     02H,D           Test if UPDATE password specified
5C15 280A       JR      Z,5C21H         Jump if not
5C17 3A6851     LD      A,(5168H)       Yes. Get new UPDATE password encode
5C1A 77         LD      (HL),A          and save in directory
5C1B 3A6951     LD      A,(5169H)       Fetch 2nd byte of encode
5C1E 23         INC     HL              Bump to 2nd byte of password
5C1F 77         LD      (HL),A          Save 2nd byte of UPDATE encode
5C20 2B         DEC     HL              Back-up for following increments
5C21 23         INC     HL              Bump to starting position of access
5C22 23         INC     HL              passwword in directory
5C23 CB4A       BIT     01H,D           Test for KILL/RENAME/READ
                                        WRITE/EXECUTE access
5C25 2809       JR      Z,5C30H         Jump if all options allowed
5C27 3A6A51     LD      A,(516AH)       Else get UPDATE password encode (LSB)
5C2A 77         LD      (HL),A          and copy to directory
5C2B 3A6B51     LD      A,(516BH)       Fetch UPDATE password encode (MSB)
5C2E 23         INC     HL              Bump to 2nd byte of password
5C2F 77         LD      (HL),A          Store MSB encode in directory
5C30 CDD64A     CALL    4AD6H           Write updated directory sector
5C33 C24C55     JP      NZ,554CH        Jump if any error during WRITE
5C36 C32D40     JP      402DH           Done. Recall SYS1, wait for command
*
*
*
5C39 21425C     LD      HL,5C42H        Address of ATTRIBUTE
                                        SPECIFICATION ERROR
5C3C CD6744     CALL    4467H           Display error message
5C3F C33040     JP      4030H           Return to SYS1 and wait for next cmd.
*
*
*
5C42            DEFB    0AH             Line feed
5C43            DEFM                    'ATTRIBUTE SPECIFICATION ERROR'
5C60            DEFB    0DH             Message terminator
*
*         PROT Parameter List for ATTRIBUTE Processing
*
5C61            DEFM                    'EXREWR  NAKIFU'
*
*                   DIR command processing
*
5C6F 010000     LD      BC,0000H        Load 16-Bit zero value to clear
5C72 ED439B5D   LD      (5D9BH),BC      A option flag (set not selected)
5C76 ED430B5D   LD      (5D0BH),BC      I option flag (set not selected)
5C7A ED43FD5C   LD      (5CFDH),BC      S option flag (set not selected)
5C7E 0E00       LD      C,00H           Default drive number is 0
5C80 7E         LD      A,(HL)          Get next char from command list
5C81 FE3A       CP      3AH             Compare with a :
5C83 2006       JR      NZ,5C8BH        Jump if no drive specified
5C85 23         INC     HL              Drive specified. Bump cmd list addr
5C86 7E         LD      A,(HL)          to drive. Get ASCII drive no.
5C87 23         INC     HL              Bump to character beyond drive no.
5C88 D630       SUB     30H             Convert drive number to binary
5C8A 4F         LD      C,A             and save in C for nucleus I/O calls
5C8B C5         PUSH    BC              Save drive for later use
5C8C 79         LD      A,C             Refetch it
5C8D C630       ADD     A,30H           and convert it to ASCII
5C8F 329E5E     LD      (5E9EH),A       Save drive no. in title message
5C92 116A5E     LD      DE,5E6AH        Address of DIR option I, S, A
5C95 CD7644     CALL    4476H           Call SYS1 to parse rest of command
5C98 21835E     LD      HL,5E83H        Address of DIR title line
5C9B CD6744     CALL    4467H           Display title line
5C9E C1         POP     BC              Reload drive number in C
5C9F C5         PUSH    BC              And preserve BC
5CA0 CD554B     CALL    4B55H           Get dir track number in D register
5CA3 1E00       LD      E,00H           Select sector 0
5CA5 210042     LD      HL,4200H        Use system buffer at 4200
5CA8 CD354B     CALL    4B35H           Read track 11, sector 0
5CAB C24C55     JP      NZ,554CH        Jump if error during read
5CAE 21D042     LD      HL,42D0H        Addr of disk name in GAT sec buf
5CB1 11A15E     LD      DE,5EA1H        Addr of disk name in 3rd title line
5CB4 010800     LD      BC,0008H        Number of bytes in diskette name
5CB7 EDB0       LDIR                    Move name from GAT buffer to 3rd line
5CB9 11AD5E     LD      DE,5EADH        Addr of date in 3rd title line
5CBC 010800     LD      BC,0008H        Number of bytes in date
5CBF EDB0       LDIR                    Move date to 3rd title line
5CC1 21A15E     LD      HL,5EA1H        Address of 3rd title line
5CC4 CD6744     CALL    4467H           Display 3rd title line
5CC7 C1         POP     BC              Restore drive no. to C
5CC8 0600       LD      B,00H           Setup for reading 1st dir sector
5CCA 110061     LD      DE,6100H        Sector buffer address
5CCD CDC14A     CALL    4AC1H           Read sector of dir file into 4200
5CD0 C24C55     JP      NZ,554CH        Jump if error during read
5CD3 2E00       LD      L,00H           Force HL to 4200. Sector buf origin
5CD5 C5         PUSH    BC              Save sector number/drive number
5CD6 010001     LD      BC,0100H        Number of bytes to move (256)
5CD9 EDB0       LDIR                    Copy buffer at 4200 to 6100
5CDB C1         POP     BC              Restore sector/drive number
5CDC 04         INC     B               Bump sector number
5CDD 78         LD      A,B             Move sector number for comparison
5CDE FE08       CP      08H             Did we read last directory sector
5CE0 20EB       JR      NZ,5CCDH        Jump if not. Read next sector
5CE2 210061     LD      HL,6100H        All sectors have been read.
                                        Display files
5CE5 3E0C       LD      A,0CH           No. of lines to display before pause
5CE7 32AB5D     LD      (5DABH),A       Initialize line counter
5CEA 0603       LD      B,03H           No. of files displayed per line
5CEC 7E         LD      A((HL)          Fetch access control byte from dir
5CED E5         PUSH    HL              Save dir addr for current file
5CEE CB67       BIT     04H,A           Test if entry is occupied
5CF0 CABF5D     JP      Z,5DBFH         If not, go get next entry
5CF3 CB7F       BIT     07H,A           Entry used. Is it primary or overflow
5CF5 C2BF5D     JP      NZ,5DBFH        Jump if overflow. Skip this
                                        entry, get next one
5CF8 CB77       BIT     06H,A           Test if user or SYSTEM file
5CFA 280A       JR      Z,5D06H         Jump if SYSTEM file
```

```
5CFC 110000     LD      DE,0000H        FFFF if S option specified
5CFF 7A         LD      A,D             Test if user specified S option
5D00 B3         OR      E               DE = 0000 if not (FFFF if so)
5D01 CABF5D     JP      Z,5DBFH         Jump if S option not selected
5D04 180C       JR      5D12H           S option selected. Go display file
5D06 CB5F       BIT     03H,A           Test if file is invisible
5D08 2808       JR      Z,5D12H         Jump if file name is displayable
5D0A 110000     LD      DE,0000H        FFFF if I option specified
5D0D 7A         LD      A,D             DE = 0000 if not, FFFF otherwise
5D0E B3         OR      E               Contents of 5D0B-5D0C modified by
                                        SYS1 if I option detected
5D0F CABF5D     JP      Z,5DBFH         Jump if I option not selected
5D12 C5         PUSH    BC              Save file's line count/drive no.
5D13 7D         LD      A,L             LSB of current entry address
5D14 C605       ADD     A,05H           Add 5 to form addr of file name
5D16 6F         LD      L,A             HL = addr of name in dir sector
5D17 0E11       LD      C,11H           Max no. chars in file name/ext
5D19 0608       LD      B,08H           Max no. chars in file name
5D1B 7E         LD      A,(HL)    .<-+  Fetch a character from name
5D1C 23         INC     HL        .  |  Bump to next character
5D1D FE20       CP      20H       .  |  Is last character a blank
5D1F 2808       JR      Z,5D29H   .  |  if yes, go display extension
5D21 CD3300     CALL    0033H     .  |  Display a char from file name
5D24 0D         DEC     C         .  |  Count 1 character displayed
5D25 10F4       DJNZ    5D1BH     .--+  Loop till name displayed
5D27 1804       JR      5D2DH           Name displayed. Go display ext
5D29 7D         LD      A,L             Name not 8 chars. Get LSB of current
5D2A 80         ADD     A,B             addr in dir buf. Add remainder of
5D2B 3D         DEC     A               bytes (-1) left in name to get
                                        address of extension
5D2C 6F         LD      L,A             Form address of extension in HL
5D2D 7E         LD      A,(HL)          Fetch 1st byte of extension
5D2E FE20       CP      20H             and compare it to a blank
5D30 2814       JR      Z,5D46H         If blank, no ext. Go blank ext area
5D32 3E2F       LD      A,2FH           ASCII /
5D34 CD3300     CALL    0033H           Display / after file name
5D37 0D         DEC     C               Dec count for total field size
5D38 0603       LD      B,03H           Max no. chars in extension
5D3A 7E         LD      A,(HL)    .<-+  Fetch a byte from ext name
5D3B 23         INC     HL        .  |  Bump to next char in extension
5D3C FE20       CP      20H       .  |  Is current char a blank
5D3E 2806       JR      Z,5D46H   .--+  Jump if yes. End of extension
                                  .  |  reached. Go get next file
5D40 CD3300     CALL    0033H     .  |  Display current char in ext
5D43 0D         DEC     C         .--+  Dec count for total field size
5D44 10F4       DJNZ    5D3AH     .<-+  Loop till extension displayed
5D46 3E20       LD      A,20H           Follow file name or extension
5D48 CD3300     CALL    0033H           With a blank
5D4B 0D         DEC     C               Dec count displayed in field
5D4C 7D         LD      A,L             Fetch LSB of current file entry
5D4D E6E0       AND     0E0H            Force LSB to beginning of file
5D4F 6F         LD      L,A             Form beginning of file addr in HL
5D50 46         LD      B,(HL)          Fetch access flag
5D51 3E53       LD      A,53H           ASCII S in case of system file
5D53 CB70       BIT     06H,B           Is it a system file
5D55 2002       JR      NZ,5D59H        Jump if yes, display S
5D57 3E20       LD      A,20H           No, replace S with a space
5D59 CD3300     CALL    0033H           Display space/S, depending on type
5D5C 3E49       LD      A,49H           ASCII I in case file is invisible
5D5E CB58       BIT     03H,B           Test if file viewable
5D60 2002       JR      NZ,5D64H        Jump if file invisible
5D62 3E20       LD      A,20H           Not invisible, replace I with space
5D64 CD3300     CALL    0033H           Display space/I,
                                        depending on attribute
5D67 E5         PUSH    HL              Save beginning addr of file entry
5D68 7D         LD      A,L             Get LSB of file entry address
5D69 C610       ADD     A,10H           Add 10 to get addr of UPDATE password
5D6B 6F         LD      L,A             Form addr of UPDATE password in HL
5D6C 5E         LD      E,(HL)          Fetch LSB of update password
5D6D 2C         INC     L               Bump to next byte in password
5D6E 56         LD      D,(HL)          Fetch MSB of update password
5D6F E5         PUSH    HL              Save address of UPDATE password
5D70 219642     LD      HL,4296H        Value for universal password
5D73 ED52       SBC     HL,DE           Test if file password protected
5D75 E1         POP     HL              Restore addr of UPDATE password
5D76 2814       JR      Z,5D8CH         Jump if file has no password
5D78 78         LD      A,B             Fetch access flags
5D79 E607       AND     07H             Isolate access control flags
5D7B 3E50       LD      A,50H           ASCII P
5D7D 200F       JR      NZ,5D8EH        Jump if not unrestricted access
5D7F 2C         INC     L               Bump to LSB of ACCESS password
5D80 5E         LD      E,(HL)          Fetch LSB of ACCESS password
5D81 2C         INC     L               Bump to MSB of ACCESS password
5D82 56         LD      D,(HL)          Fetch MSB of ACCESS password
5D83 219642     LD      HL,4296H        Universal password code
5D86 ED52       SBC     HL,DE           Is there any ACCESS password
5D88 3E50       LD      A,50H           ASCII P in case there is a password
5D8A 2002       JR      NZ,5D8EH        Go if ACCESS password. Display P
5D8C 3E20       LD      A,20H           No password. Replace P with space
5D8E CD3300     CALL    0033H           Disp. P/space,
                                        depending on permission
5D91 E1         POP     HL              Restore beginning addr of file entry
*
*               Blank fill remainder of field
*
5D92 3E20       LD      A,20H     .<-+  A = ASCII space
5D94 CD3300     CALL    0033H     .  |  Display a space
5D97 0D         DEC     C         .  |  Dec count for total field size
5D98 20F8       JR      NZ,5D92H  .--+  Loop till field filled
5D9A 110000     LD      DE,0000H        FFFF if A option specified
5D9D 7A         LD      A,D             Combine LSB and MSB
5D9E B3         OR      E               Of A option byte. Changed by SYS1
                                        if A option specified
5D9F C1         POP     BC              Restore files/line count, drive no.
5DA0 C4F15D     CALL    NZ,5DF1H        If A option, go display file size
5DA3 101A       DJNZ    5DBFH           Count 1 file displayed. Jump if
                                        room for more files this line
5DA5 3E0D       LD      A,0DH           Else skip to next line
5DA7 CD3300     CALL    0033H           By displaying a carriage return
5DAA 3E00       LD      A,00H           Get max no. of lines per screen
5DAC 3D         DEC     A               Decrement it
5DAD 2005       JR      NZ,5DB4H        Go if room for more lines this screen
5DAF CD4900     CALL    0049H           Else wait for operator reply
5DB2 3E0C       LD      A,0CH           Max. number of lines per screen
5DB4 32AB5D     LD      (5DABH),A       reset line/screen counter
5DB7 CD1555     CALL    5515H           Test for BREAK key
5DBA C2DB5D     JP      NZ,5DDBH        Jump if BREAK key active
5DBD 0603       LD      B,03H           Reload no. of files/line
counter
5DBF E1         POP     HL              Restore beginning addr of
                                        current directory entry
5DC0 24         INC     H               Bump to next sector
5DC1 7C         LD      A,H             Test if end of sector list reached
*
*       Loop control for DIR. All directory sectors are read
*       into 8 sector buffers beginning at 6100, 6200, . . .
*       before the main loop is entered. Each pass through the
*       loop processes one file entry from a sector buffer. At
*       the end of the loop the MSB of the file entry address
*       is incremented by 100 giving the address of the
*       corresponding entry in the next sector. When the end
*       of the sector buffers is reached, the LSB of the file
*       entry address is incremented by 20 to get the address
*       of the next entry in the last sector. Then the MSB
*       byte of the next entry address is set to 61 giving the
*       address of the next file entry in the first sector,
*       etc.
*
5DC2 FE69       CP      69H             Have 8 sectors been processed?
5DC4 C2EC5C     JP      NZ,5CECH        If not go process entry from next sec
5DC7 2661       LD      H,61H           Reset MSB of buf pointer to 1st sec
5DC9 7D         LD      A,L             Fetch LSB of beginning address
                                        for last entry
5DCA C620       ADD     A,20H           Add 20 to compute beginning addr
5DCC 6F         LD      L,A             of next entry. form addr in HL
5DCD D2EC5C     JP      NC,5CECH        Jump if all entries not processed
5DD0 CDE15D     CALL    5DE1H           Zero sector buffers
                                        to protect passwords
5DD3 3E0D       LD      A,0DH           Skip to next line
5DD5 CD3300     CALL    0033H           By displaying a carriage return
5DD8 C32D40     JP      402DH           Ret to SYS1 to get next command
5DDB CDE15D     CALL    5DE1H           Zero sector buffer for security
5DDE C33040     JP      4030H           Ret to SYS1 for next command
*
*
*
5DE1 210061     LD      HL,6100H        Beginning addr of sector buffer
5DE4 0600       LD      B,00H           Sector index
5DE6 70         LD      (HL),B          Zero a byte of a sector
5DE7 2C         INC     L               Bump to next byte of sector
5DE8 20FC       JR      NZ,5DE6H        Loop till one sector zeroed
5DEA 24         INC     H               Bump to next sector address
5DEB 7C         LD      A,H             Test if all buffers cleared
5DEC FE69       CP      69H             End of sector buffers reached
5DEE 20F6       JR      NZ,5DE6H        Jump if not. Zero next sector
5DF0 C9         RET                     Yes, return to caller
*
*       'A' Option of DIR Command. Display File Size.
*
5DF1 E5         PUSH    HL              Save beginning addr of file entry
5DF2 2C         INC     L               Bump by 3
5DF3 2C         INC     L               to get address of
5DF4 2C         INC     L               EOF sector
5DF5 7E         LD      A,(HL)          Fetch EOF byte offset
5DF6 32695E     LD      (5E69H),A       Save in temp. storage area
5DF9 2C         INC     L               Bump to logical record length in dir
5DFA E5         PUSH    HL              Save address of LRL
5DFB 6E         LD      L,(HL)          Fetch logical record length
5DFC 7D         LD      A,L             Move it to A so it can
5DFD 321F5E     LD      (5E1FH),A       be saved in temp. storage
5E00 D601       SUB     01H             Sub 1 to force CARRY if LRL = 256 (0)
5E02 3E00       LD      A,00H           Then add CARRY to a
5E04 CE00       ADC     A,00H           Byte of zeroes. This yields a
5E06 67         LD      H,A             16 Bit value for LRL (01-0100)
5E07 11BB5E     LD      DE,5EBBH        Addr of LRL in display line
5E0A CD6C60     CALL    606CH           Convert to ASCII, save in display
5E0D E1         POP     HL              Restore address of LRL
5E0E 7D         LD      A,L             Fetch LSB of address
5E0F C610       ADD     A,10H           and add 10 to addr of byte
                                        containing EOF sector no.
5E11 6F         LD      L,A             Reform addr of EOF sector in HL
5E12 7E         LD      A,(HL)          Fetch LSB of ending sector no.
5E13 32685E     LD      (5E68H),A       Save in temp. storage
5E16 2C         INC     L               Bump to MSB of EOF sector
5E17 7E         LD      A,(HL)          Fetch MSB of ending sector no.
5E18 32675E     LD      (5E67H),A       Save in temp. storage
5E1B 21675E     LD      HL,5E67H        Addr of ending sector no. in binary
5E1E 0E00       LD      C,00H           Load logical record length.
                                        Instr. modified at 5DFD
5E20 CDA760     CALL    60A7H           Divide no. of records by record size
5E23 EB         EX      DE,HL           Quotient to HL
5E24 11C75E     LD      DE,5EC7H        Addr of ending sec no. in display
5E27 CD6C60     CALL    606CH           Convert ending sector no. to ASCII
5E2A E1         POP     HL              Restore address of LRL
5E2B CD3E5E     CALL    5E3EH           Go compute no. grans assigned to file
5E2E EB         EX      DE,HL           Move granule count to HL
5E2F 11D45E     LD      DE,5ED4H        Addr of no. grans in display line
5E32 CD6C60     CALL    606CH           Convert gran count to decimal ASCII
5E35 21B75E     LD      HL,5EB7H        Address of LRL display line
5E38 CD6744     CALL    4467H           Display LRL/EOF offset and # grans
5E3B 0601       LD      B,01H           Replace line counter with 1 because
5E3D C9         RET                     only one file/line with this
                                        display. Return to caller
*
*
*
5E3E 110000     LD      DE,0000H        Zero accumulator
5E41 7D         LD      A,L             Fetch LSB of LRL
5E42 C616       ADD     A,16H           Add 16 to position to GAP's
5E44 6F         LD      L,A             Form address of GAP's in HL
```

```
5E45 7E        LD      A,(HL)        .<-Fetch 1st byte of a GAP
5E46 2C        INC     L             .  Bump to granule count
5E47 FEFE      CP      0FEH          .  Test for end of GAP's or overflow
5E49 300C      JR      NC,5E57H      .  Jump if end or overflow
5E4B 7E        LD      A,(HL)        .  Load granule count
5E4C 2C        INC     L             .  Bump to next GAP
5E4D E61F      AND     1FH           .  Isolate granule count
5E4F 3C        INC     A             .  Compute true granule count
5E50 83        ADD     A,E           .  Form sum in DE
5E51 5F        LD      E,A           .  Update LSB of sum
5E52 30F1      JR      NC,5E45H      .-Jump if LSB has not oveflowed
5E54 14        INC     D             .  else bump MSB of gran count
5E55 18EE      JR      5E45H         .--Loop till end of GAP's found
5E57 C0        RET     NZ            Exit if FF found (end of
GAP's)
5E58 46        LD      B,(HL)        Get pointer to overflow entry
5E59 78        LD      A,B           Move to A
5E5A E607      AND     07H           and isolate the sector number
5E5C C661      ADD     A,61H         Then form addr of sector buffer
5E5E 67        LD      H,A           and move MSB of addr to H
5E5F 78        LD      A,B           Now compute offset within sector
5E60 E6E0      AND     0E0H          Isolate offset to file entry
5E62 C616      ADD     A,16H         and add offset to GAP's
5E64 6F        LD      L,A           Form addr of overflow GAP's in HL
5E65 18DE      JR      5E45H         Continue till end of GAP's found
*
*
*
5E67 DEFB      0                     MSB of EOF sector
5E68 DEFB      0                     LSB of EOF sector
5E69 DEFB      0                     Holds EOF byte offset
*
*                  DIR Option Text List
*
5E6A           DEFM    'A     '      Option
5E71           DEFW    5D9BH         Code address for A option
5E72           DEFM    'I     '      Option
5E78           DEFW    5D0BH         Code address for I option
5E7A           DEFW    'S     '      Option
5E81           DEFW    5CFDH         Code address for S option
5E82           DEFB    00H           List terminator
*
*                  DIR Screen Displays
*
5E83           DEFB    1CH           Clear screen
5E84           DEFB    1FH           Home cursor
5E85           DEFM    'FILE DIRECTORY  --- DRIVE X'
5E9F           DEFB    C5H           Compression code for 5 blanks
5EA0           DEFB    03H           Message terminator
5EA1           DEFM    'NNNNNNNN -- MM/DD/YY'
5EB5           DEFB    0AH           Line feed
5EB6           DEFB    0DH           Message terminator
5EB7           DEFM    'LRL=      / END=      / SIZE'
5EDF           DEFB    03H           Message terminator
*
*                  FREE Command Processing
*
5EE0 0E00      LD      C,00H         Default beginning drive number
5EE2 3E03      LD      A,03H         Illegal character to local DCB
5EE4 115155    LD      DE,5551H      Address of local DCB
5EE7 12        LD      (DE),A        Store illegal character so OPEN
                                     will return immediately
5EE8 CD2444    CALL    4424H         OPEN dummy file
                                     (load SYS2 into 4E00)
5EEB C5        PUSH    BC            Save drive number
5EEC CDFD50    CALL    50FDH         See if drive available
                                     (use SYS2 subroutine)
5EEF 2067      JR      NZ,5F58H      Jump if drive not available
5EF1 79        LD      A,C           Fetch current drive number
5EF2 C630      ADd     A,30H         Convert drive number to ASCII
5EF4 32685F    LD      (5F68H),A     and move it to display line
5EF7 CDF04A    CALL    4AF0H         Read GAT sector for drive
5EFA C24C55    JP      NZ,554CH      Jump if error GAT sector read
5EFD 21D04D    LD      HL,4DD0H      Addr of disk name in GAT sector
5F00 116E5F    LD      DE,5F6EH      Addr of disk name in display
5F03 010800    LD      BC,0008H      Number of bytes in disk name
5F06 EDB0      LDIR                  Move name from GAT sector to display
5F08 11795F    LD      DE,5F79H      Addr of creation date on display
5F0B 010800    LD      BC,0008H      Number of bytes in date
5F0E EDB0      LDIR                  Move date from GAT sector to display
5F10 21004D    LD      HL,4D00H      Start addr of GAT table in GAT sec
5F10 21004D    LD      HL,4D00H      Start addr of GAT table in GAT sec
5F13 110000    LD      DE,0000H      Zero accumulator. Will hold
                                     number of free grans
5F16 0623      LD      B,23H         Max no. of GAT entries to examine
5F18 7E        LD      A,(HL)        Fetch a GAT entry
5F19 37        SCF                   Set CARRY so 1 is forced to bit 7
5F1A 1F        RRA                   Bit 0 (granule) to carry
5F1B 3801      JR      C,5F1EH       Go if gran 0, current track assigned
5F1D 13        INC     DE            If not, count 1 free granule
5F1E FEFF      CP      0FFH          Test if bit 1 (gran 1) available
5F20 20F7      JR      NZ,5F19H      Jump if granule 1 available
5F22 2C        INC     L             Bump to next track in GAT
5F23 10F3      DJNZ    5F18H         Loop till all tracks tested
*
*
*
5F25 EB        EX      DE,HL         Count of available grans to HL
5F26 11905F    LD      DE,5F90H      Array addr to save ASCII value
5F29 CD6C60    CALL    606CH         Convert count to ASCII and save
5F2C C1        POP     BC            Restore current drive no. to C
5F2D C5        PUSH    BC            and save for later use
5F2E CD2E51    CALL    512EH         Read HIT sector (use SYS2 subroutine)
*
*
```

```
*       Prepare to Compute the Number of Files Assigned to
*            Diskette by Counting HIT Indices
*
*
5F31 C24C55    JP      NZ,554CH      Jump if error during read
5F34 21404D    LD      HL,4D40H      Start of HIT area for user files
5F37 110000    LD      DE,0000H      Clear accumulator
5F3A 7E        LD      A,(HL)        Fetch a HIT index
5F3B B7        OR      A             Set status flags
5F3C 2001      Jr      NZ,5F3FH      .--Go if file assigned this slot
5F3E 13        INC     DE            .  Else count 1 space available
5F3F 2C        INC     L             .<-Bump to next slot in HIT
5F40 7d        LD      A,L           .  Prepare to test for
5F41 E6E7      AND     0E7H          .  end of slot range
5F43 BD        CP      L             .  Reached end of a slot range?
5F44 28F4      JR      Z,5F3AH       .  If not, go continue searching
5F46 C620      ADD     A,20H         .  Yes, bump to next slot area
5F48 6F        Ld      L,A           .  Form next HIT address in HL
5F49 20EF      JR      NZ,5F3AH      .  Loop if end of HIT not reached
5F4B EB        EX      DE,HL         Free HIT slots to HL for conversion
5F4C 11835F    LD      DE,5F83H      5 byte array addr for ASCII HIT slots
5F4F CD6C60    CALL    606CH         Convert free slots to ASCII decimal
5F52 21625F    LD      HL,5F62H      Addr of display line DRIVE X-- ...
5F55 CD6744    CALL    4467H         Display title line with values
5F58 C1        POP     BC            Restore drive no. to C register
5F59 0C        INC     C             Bump to next drive
5F5A 79        LD      A,C           Move drive to A for
5F5B FE04      CP      04H           Compare. Test if all drives done
5F5D 208C      JR      nZ,5EEBH      Jump if more drives to process
5F5F C32D40    JP      402DH         Done, ret to SYS1 for next command
*
*                FREE display messages
*
5F62           DEFM    'DRIVE X --  '
5F6E           DEFM    'NNNNNNNN  MM/DD/YY        FILES'
5F8F           DEFM    '       GRANS'
*
5F9B           DEFB    0DH           Message terminator
*
*                RENAME Command Processing
*
5F9C 115155    LD      DE,5551H      Load DCB address
5F9F CD1C44    CALL    441CH         SYS1/code 4, copy file name to DCB
5FA2 C23A55    JP      NZ,553AH      Jump if illegal chars in name
5FA5 1A        LD      A,(DE)        Fetch 1st character of name
5FA6 FE2A      CP      2AH           Test for special system file (*)
5FA8 CA3A55    JP      Z,553AH       Error if found (file spec. required)
5FAB CD2444    CALL    4424H         OPEN file, load SYS2 at 4E00
5FAE C24C55    JP      NZ,554CH      Jump if file not present
5FB1 117155    LD      DE,5571H      Local DCB addr for 2nd file name
5FB4 CD1C44    CALL    441CH         SYS1/code 4, Copy file name to DCB
5FB7 C23A55    JP      NZ,553AH      Jump if illegal chars in new name
5FBA 3A5255    LD      A,(5552H)     Get access flags for file to RENAME
5FBD E607      AND     07H           Isolate access permission
5FBF FE03      CP      03H           Test for RENAME permission
5FC1 3E25      LD      A,25H         Error code for not allowed
5FC3 D24C55    JP      NC,554CH      Go if RENAME permission required
*
*       Scan file name looking for a drive specification.
*           If one is found an error has occurred.
*
5FC6 217155    LD      HL,5571H      .<-Addr of DCB with new file name
5FC9 7E        LD      A,(HL)        .  Get a character from file name
5FCA FE3A      CP      3AH           .  Test for : (drive spec)
5FCC 285A      JR      Z,6028H       .  Go if : found (error)
5FCE FE20      CP      20H           .--Test for end of name
5FD0 3803      JR      C,5FD5H       .  Jump if end of file name
5FD2 23        INC     HL            .--Else bump to next character
5FD3 18F4      JR      5FC9H         .<-and loop till end of name
5FD5 363A      LD      (HL),3AH      Then add drive specification
5FD7 23        INC     HL            Bump to next position in DCB
5FD8 3A5755    LD      A,(5557H)     Fetch drive number (in binary)
5FDB C630      ADD     A,30H         Convert to ASCII
5FDD 77        LD      (HL),A        and save in DCB after new name
5FDE 23        INC     HL            Bump to first pos. past drive spec.
5FDF 3603      LD      (HL),03H      and store a terminator
5FE1 117155    LD      DE,5571H      Addr of DCB with new file name
5FE4 CD2444    CALL    4424H         OPEN new file name
5FE7 CA4E60    JP      Z,604EH       Go if name already exists (error)
5FEA FE18      CP      18H           Test status for other error
5FEC C24C55    JP      NZ,554CH      Jump if some other error occurred
5FEF ED4B5755  LD      BC,(5557H)    Drive no. to C, dir sector # to B
5FF3 C5        PUSH    BC            Save drive number
5FEF ED4B5755  LD      BC,(5557H)    Drive no. to C, dir sector # to B
5FF3 C5        PUSH    BC            Save drive number
5FF4 CDC14A    CALL    4AC1H         Read dir sector for file to rename
5FF7 C24C55    JP      NZ,554CH      Go if error during dir read
5FFA 54        LD      D,H           Form addr for file entry in DE
5FFB 7D        LD      A,L           by
5FFC C605      ADD     A,05H         adding 5 to the address in HL
5FFE 5F        LD      E,A           and moving it to DE
5FFF 215D51    LD      HL,515DH      Addr of SYS2 DCB with new file name
6002 010B00    LD      BC,000BH      Max no. of chars in new file name
6005 EDB0      LDIR                  Copy new name from buffer in SYS2
                                     to directory file name area
6007 C1        POP     BC             Restore dir sector/drive no.
6008 CDD64A    CALL    4AD6H         Rewrite directory sector
600B C24C55    JP      NZ,554CH      Jump if error during write
600E CD2E51    CALL    512EH         Use SYS2 subroutine to read HIT
6011 C24C55    JP      NZ,554CH      Jump if error while reading HIT
6014 54        LD      D,H           H = MSB of HIT sector buffer.
                                     B = directory sector number
6015 58        LD      E,B           DE = addr of HIT slot for this file
6016 C5        PUSH    BC            Save directory sector/drive number
6017 215D51    LD      HL,515DH      Addr of SYS2 buffer with new name
601A CD9B50    CALL    509BH         Compute hash code for new file
601D C1        POP     BC            Restore dir sector/drive number
601E 12        LD      (DE),A        Save new hash code
601F CD4151    CALL    5141H         SYS2 subroutine to write updated HIT
6022 C24C55    JP      NZ,554CH      Jump if error while writing HIT
```

```
6025 C32D40     JP      402DH       Done, load SYS1 for next command
*
*
*
6028 213160     LD      HL,6031H    Address of DRIVE SPECIFICATION
                                    ILLEGAL message
602B CD6744     CALL    4467H       Display error message
602E C33040     JP      4030H       Recall SYS1 wait nor next command
*
*                   RENAME Error Messages
*
6031            DEFB    0AH         Line feed
6032            DEFM    'DRIVE SPECIFICATION ILLEGAL'
604D            DEFB    0DH         Message terminator
*
*
*
604E 215760     LD      HL,60=7H    Address of DUPLICATE FILE NAME msg.
6051 CD6744     CALL    4467H       Display error message
6054 C33040     JP      4030H       Recall SYS1 and wait for next cmd.
*
*
*
6057            DEFB    0AH         Line feed
6058            DEFM    'DUPLICATE FILE NAME'
606B            DEFB    0DH         Message terminator
*
*     Convert value in HL to decimal ASCII. On entry DE =
*     address of 5-byte array where result is to be stored
*
606C 3E20       LD      A,20H       Pad character
606E 011027     LD      BC,2710H    BC = 10,000 dec.
6071 CD8C60     CALL    (608CH      Divide by 10,000. Save quotient
6074 01E803     LD      BC,03E8H    BC = 1,000 dec.
6077 CD8C60     CALL    608CH       Divide by 1,000. Save quotient
607A 016400     LD      BC,0064H    BC = 100 dec.
607D CD8C60     CALL  ( 608CH       Divide by 100. Save quotient
6080 010A00     LD      BC,000AH    BC = 10 dec.
6083 CD8C60     CALL    608CH       Divide by 10. Save quotient
6086 7D         LD      A,L         Get LSD of value
6087 C630       ADD     A,30H       and convert it to ASCII
6089 12         LD      (DE),A      and save
608A 13         INC     DE          Bump to address of next
                                    byte in ASCII list
608B C9         RET                 Return to caller
*
*
*
608C D5         PUSH    DE          Save address where ASCII
                                    value is to be stored
608D 5F         LD      E,A         Save pad character
608E 16FF       LD      D,0FFH      D is quotient. Initialize as -1
6090 AF         XOR     A           Clear CARRY
6091 14         INC     D         .<- Bump quotient for division
6092 ED42       SBC     HL,BC     . | Simulate division by compound
                                  . | subtraction
6094 30FB       JR      NC,6091H  .-- Loop till divisor greater
                                  .   than dividend
6096 09         ADD     HL,BC       Make dividend (remainder) positive
6097 7B         LD      A,E         Pad character
6098 42         LD      B,D         B = quotient for current division
6099 D1         POP     DE          Restore storage address for ASCII
                                    equivalent of value
609A 12         LD      (DE),A      Save pad character. Will be replaced
                                    w/quotient if remainder non-zero
609B 04         INC     B           Set Status flag
609C 05         DEC     B           for remainder
609D 2806       JR      Z,60A5H     Jump if no remainder
                                    (division was exact)
609F 78         LD      A,B         Quotient for last division
60A0 C630       ADD     A,30H       Convert it to ASCII
60A2 12         LD      (DE),A      and save in ASCII list
60A3 3E30       LD      A,30H       Set next pad char. to ASCII 0
60A5 13         INC     DE          Bump to next address in ASCII list
60A6 C9         RET                 and return to caller
*
*
*
60A7 110000     LD      DE,0000H    Clear accumulator
60AA 79         LD      A,C         Fetch divisor
60AB B7         OR      A           Set status flag for division
60AC 281B       JR      Z,60C9H     Jump if divisor zero
60AE AF         XOR     A           Clear CARRY
60AF 0603       LD      B,03H       Number of bytes in dividend
60B1 C5         PUSH    BC        .<-- Save number of bytes
                                  . |  remaining to be divided
60B2 E5         PUSH    HL        . |  Save addre{s of current
                                  . |  byte to divide
60B3 66         LD      H,(HL)    . |  Fetch a byte to divide
60B4 0608       LD      B,08H     . |  number of bits to test in byte
60B6 EB         EX      DE,HL     .<-| so we can shift DE left 1 bit.
                                  . |  Use addition.
60B7 29         ADD     HL,HL     . |  Left shift quotient
60B8 EB         EX      DE,HL     . |  Restore quotient to DE
60B9 CB04       RLC     H         . |  Most sig. bit of byte to CARRY
60BB 17         RLA               . |  Then shift it into A.
                                  . |  Combine with other bits
60BC B9         CP      C         . |  from dividend. Compare to divisor
60BD 3802       JR      C,60C1H   .-|- Jump if dividend less
                                  . | |  than divisor
60BF 91         SUB     C         . | |  Adjust dividend so it is less
                                  . | |  than divisor
60C0 13         INC     DE        . | |  Bump quotient
60C1 10F3       DJNZ    60B6H     .-|-|- Loop till all bits
                                  . | |  in dividend tested
60C3 E1         POP     HL        . | |  Restore address of current
                                  . | |  byte
60C4 23         INC     HL        . | |  Bump to next byte of dividend
60C5 C1         POP     BC        . | |  Restore quotient of
                                  . | |  bytes divided
60C6 10E9       DJNZ    60B1H     .-|-|- Loop till 3 bytes divided
60C8 C9         RET               . | |  Return to caller. DE = quotient
60C9 56         LD      D,(HL)    .<| |  Divisor was zero. Return 2 most
60CA 23         INC     HL        . | |  significant digits
60CB 5E         LD      E,(HL)    . | |  of dividend as quotient
60CC C9         RET               . | |  Return to caller
*                                   | |
*                                   | |
*                                   | |
60CD 218B47     LD      HL,478BH    | |  Address of WRITE routine in nucleus
60D0 2803       JR      Z,60D5H     | |  Go clear verify flag if (OFF) option
60D2 AF         XOR     A           | |  Zero A, clear status flags
60D3 3C         INC     A           | |  Load 1 into A (auto verify writes)
60D4 B7         OR      A           | |  Set status flags non-zero
60D5 325E48     LD      (485EH),A   | |  Save verify flag in nucleus
60D8 C9         RET                 | |  Return to caller
```

```
;*********************************************************
;*                                                       *
;*                        BOOTSYS                        *
;*                     System Loader                     *
;*                                                       *
;*********************************************************


4200 00        NOP             BOOT/SYS flag. Used by BACKUP
4201 FE11      CP      11H     Directory track number
4203 F3        DI              Disk I/O timing must
                               not be interrupted
4204 31FC41    LD      SP,41FCH  Initialize stack pointer address
4207 21E242    LD      HL,42E2H  Address of "CLS" video routine
420A CD9A42    CALL    429AH     Issue video commands
420D 3E01      LD      A,01H     Unit select mask for drive 0
420F 32E137    LD      (37E1H),A Select drive 0
4212 3A0242    LD      A,(4202H) Fetch directory track number
4215 57        LD      D,A       Put it in the D register
4216 1E04      LD      E,04H     Put SYS0 directory no. in E
4218 01004D    LD      BC,4D00H  Local buffer address
421B CDAA42    CALL    42AAH     Get SYS0 dir entry (trk 11/sec 4)
421E 2070      JR      NZ,4290H  Jump if any error during READ
4220 3A004D    LD      A,(4D00H) Get 1st byte of SYS0 dir entry
4223 E610      AND     10H       Isolate entry occupied flag
4225 21E542    LD      HL,42E5H  Address of DISK ERROR message
4228 2869      JR      Z,4293H   If no entry then go display NO SYSTEM
;
;   Initialize alternate DE and BC registers for get next
;   byte routine from sector buffer at 4275. DE contains
;   the track and sector number for the next sector to be
;   read. BC contains the current buffer address. This code
;   initializes BC to 4DFF which signals an empty sector
;   buffer.
;
422A D9        EXX             Get alternate register set
422B 2A164D    LD      HL,(4D16H)  Fetch 1st GAP for SYS0/SYS
422E 55        LD      D,L       D = track number for SYS0
422F 7C        LD      A,H       A = byte count, sector offset
4230 07        RLCA              Isolate bit 5 (sector offset)
4231 07        RLCA              and position it
4232 07        RLCA              over bit 0 so we can
4233 E607      AND     07H       isolate bit 5 (bits 6/7 are zero)
4235 67        LD      H,A       Put in H to multiply micro-style
4236 07        RLCA              Sector offset bit * 2
4237 07        RLCA              times 4
4238 84        ADD     A,H       times 5 gives 1st sector no. (0/5)
4239 5F        LD      E,A       Starting sector no. to E
423A 01FFAD    LD      BC,4DFFH  Signal empty buffer to force read
                                 on next buffer access
423D D9        EXX               Switch back to main register set
;
;               ----  Main Loop  ----
;
423E CD7542    CALL    4275H     Get next byte from sector buffer.
                                 Must be a loader code
4241 3D        DEC     A         Is it code 1? (load data follows)
4242 2017      JR      NZ,425BH  Jump if not control code 1
4244 CD7542    CALL    4275H     else get no. of bytes to load
4247 47        LD      B,A       Put load count in B for DJNZ instr.
4248 CD7542    CALL    4275H     Get LSB of load address
424B 6F        LD      L,A       Form load address in HL
424C 05        DEC     B         Count LSB as a byte loaded
424D CD7542    CALL    4275H     Get MSB of load address and
4250 67        LD      H,A       Form load address in HL
4251 05        DEC     B         Count MSB as a byte loaded
4252 28EA      JR      Z,423EH   Go if no more bytes to load
4254 CD7542    CALL    4275H     Get a byte of load data
4257 77        LD      (HL),A    Save it at current load address
4258 23        INC     HL        Bump load address and
4259 18F6      JR      4251H     loop till byte count exhausted
425B 3D        DEC     A         Test for code 2 (transfer addr flag)
425C 280B      JR      Z,4269H   Jump if control code 2
425E CD7542    CALL    4275H     Not code 1 or 2 so ignore it and
4261 47        LD      B,A       get the byte count (bytes to discard)
4262 CD7542    CALL    4275H     Fetch a byte to discard
4265 10FB      DJNZ    4262H     Go get another till all discarded
4267 18D5      JR      423EH     Go get next control code
4269 CD7542    CALL    4275H     Code 2. Skip over byte count
426C CD7542    CALL    4275H     Fetch LSB of transfer address
426F 6F        LD      L,A       Move it to L
4270 CD7542    CALL    4275H     Fetch MSB of transfer address
4273 67        LD      H,A       Form transfer addresss in HL and
4274 E9        JP      (HL)      jump (entry point for SYS0/SYS
;
;       ----  Gets Next Byte From Sector Buffer  ----
;
4275 D9        EXX               Switch to alternate register set
4276 0C        INC     C         Bump buffer addr to next byte
4277 2014      JR      NZ,428DH  Jump if sector buffer not empty
4279 C5        PUSH    BC        Save buf addr, prepare to read sector
427A 3E01      LD      A,01H     Unit select mask for drive 0
427C 32E137    LD      (37E1H),A Re-select 0 to renew MOTOR ON cycle
427F CDAA42    CALL    42AAH     Read 1 sector (trk/sec no. in DE)
4282 200C      JR      NZ,4290H  Jump if any error during read
4284 C1        POP     BC        Restore sector buffer address
4285 1C        INC     E         Bump sector number, then
4286 7B        LD      A,E       test to see if the last sector
4287 D60A      SUB     0AH       of the current track has been read
4289 2002      JR      NZ,428DH  Jump if it hasn't
428B 5F        LD      E,A       else reset sector no. to zero
428C 14        INC     D         and bump current track number
428D 0A        LD      A,(BC)    Get next byte from buffer (last byte
                                 if 428D not jumped to by 4277)
428E D9        EXX               Switch back to main register set
428F C9        RET               and return to caller
;
;
;
4290 21F142    LD      HL,42F1H  Address of DISK ERROR message
4293 CD9A42    CALL    429AH     Display error message
4296 CD4000    CALL    0040H     Then wait for keyboard input
                                 (allow operator time to
                                 change diskette)
4299 76        HALT              Always causes RESET.
;
;
;
429A E5        PUSH    HL        Save address of text message
429B 7E        LD      A,(HL)    Fetch a character from message
429C FE03      CP      03H       Test for end of message
429E 2808      JR      Z,42A8H   Jump if end of message
42A0 CD3300    CALL    0033H     Not end of message, display char.
42A3 23        INC     HL        Bump to next character in message
42A4 FE0D      CP      0DH       Test for end of message
42A6 20F3      JR      NZ,429BH  Jump if not end of message
42A8 E1        POP     HL        Restore beginning of mess. addr.
42A9 C9        RET               Return to caller
42AA C5        PUSH    BC        Save caller's BC (sec buf addr)
42AB CDB242    CALL    42B2H     Read sector specified in DE
42AE E1        POP     HL        Caller's BC reg (sec buf addr)
42AF C8        RET     Z         Return if no disk error
42B0 44        LD      B,H       Disk error, retry operation
42B1 4D        LD      C,L       after restoring caller's BC
42B2 ED53EE37  LD      (37EEH),DE  Load track/sector regs in controller
42B6 21EC37    LD      HL,37ECH  Addr of stat/cmd registers for disk
42B9 361B      LD      (HL),1BH  Issue SEEK command
42BB F5        PUSH    AF        Delay so controller
42BC F1        POP     AF        has time to respond
42BD F5        PUSH    AF        to the command
42BE F1        POP     AF        before we ask for the status
42BF 7E        LD      A,(HL)    Fetch controller status
42C0 0F        RRCA              Controller busy flag to carry flag
42C1 38FC      JR      C,42BFH   Loop till controller not busy
42C3 3688      LD      (HL),88H  Give controller READ SECTOR command
42C5 D5        PUSH    DE        Save track/sector number
42C6 11EF37    LD      DE,37EFH  Addr of controller's data register
42C9 C5        PUSH    BC        Delay for controller, so
42CA C1        POP     BC        It can react to last command
42CB 180B      JR      42D8H     Simulate CALL 42D0 to get data
                                 byte (or OP done status bit)
42CD 0F        RRCA              Controller busy status to carry
42CE 300A      JR      NC,42DAH  Go if OP done (not busy status)
42D0 7E        LD      A,(HL)    Get status
42D1 CB4F      BIT     01H,A     Test DATA READY status bit
42D3 28F8      JR      Z,42CDH   Jump if data not available
42D5 1A        LD      A,(DE)    Read a data byte from controller
42D6 02        LD      (BC),A    Save in sector buffer
42D7 03        INC     BC        Bump sector buffer address
42D8 18F6      JR      42D0H     Loop till controller not busy
42DA 7E        LD      A,(HL)    Fetch controller status
42DB E65C      AND     5CH       Isolate error flags
42DD D1        POP     DE        Restore track/sector address
42DE C8        RET     Z         Ret if no error. If error occurred,
42DF 36D0      LD      (HL),0D0H stop controller with force interrupt
42E1 C9        RET               and then return to caller
;
;
;
42E2 1C        DEFB    1CH       Clear screen
42E3 1F        DEFB    1FH       HOME cursor
42E4 03        DEFB    03H       Message terminator
42E5 17        DEFB    17H       Select 32 characters/line
42E6 E8        DEFB    E8H       Compression code for 40 blanks
42E7 4E        DEFM    'NO SYSTEM'
42F0 0D        DEFB    0DH       Message terminator
42F1 17        DEFB    17H       Select 32 characters/line
42F2 E8        DEFB    E8H       Compression code for 40 blanks
42F3 44        DEFM    'DISK ERROR'
42FD 0D        DEFB    0DH       Message terminator
42FE EB        DEFB    0EBH      Not used
42FF 5F        DEFB    05FH      Not used
```

List of Illustrations

notes

Index

*** C ***

*** D ***

*** E ***

*** T ***

*** U ***

*** V ***

*** W ***

*** X ***

*** Y ***

*** Z ***

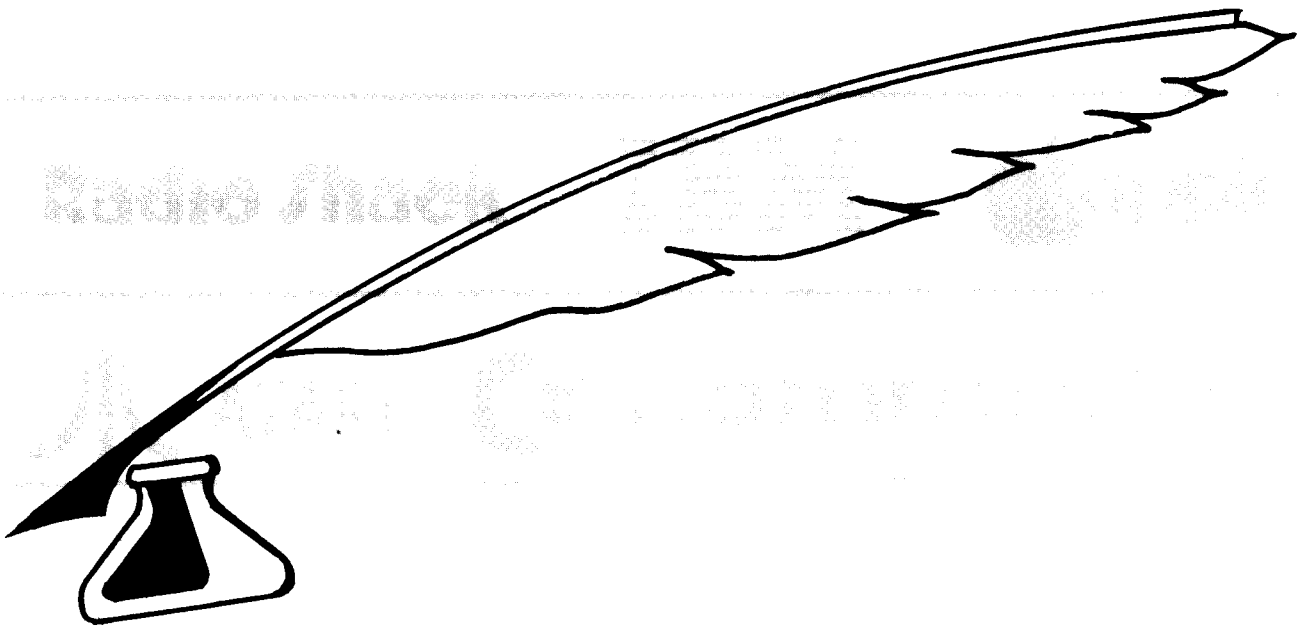
notes

notes

notes

notes

WANTED



Book & Software AUTHORS

IJG INC. has become a world wide recognized leader in computer publishing. We take pride in publishing only the best in computer oriented books and software. If you have an idea, and really know your subject, we would like to talk with you.

Qualifying manuscripts once submitted, will be read and evaluated by our professional editorial staff (who are themselves published authors), and a few selected writers will be invited in for a personal evaluation of their work.

Contact Mr. Harvard Pennington or Mr. David Moore.

IJG

1953 West
11th Street
Upland,CA
91786 (714)
946-5805

JG Computer Books & Software

Microsoft trademark Microsoft Corporation
Apple trademark Apple Computer Inc.
TRS-80 trademark TANDY Corporation
Electric Pencil © 1981 Michael Shrayer

BOOKS

TRS-80 Disk & Other Mysteries. H.C. Pennington. The "How to" book of Data Recovery. 128 pages. **$22.50**

Microsoft Basic Decoded & Other Mysteries. James Farvour. The Complete Guide to Level II Operating Systems & BASIC. 312 pages **$29.95**

The Custom TRS-80 & Other Mysteries. Dennis Bathory Kitsz. The Complete Guide to Customizing TRS-80 Software & Hardware. 336 pages **$29.95**

BASIC Faster & Better & Other Mysteries. Lewis Rosenfelder. The Complete Guide to BASIC Programming Tricks & Techniques. 290 pages **$29.95**

Electric Pencil Operators Manual. Michael Shrayer. Electric Pencil Word Processing System Manual. 123 pages. **$24.95**

The Custom Apple. Winfried Hofacker & Ekkehard Floegel. The Complete Guide to Customizing the Apple Software and Hardware. 190 pages **$24.95**

Machine Language Disk I/O & Other Mysteries. Michael D. Wagner. A Guide to Machine Language Disk I/O for the TRS-80 Models I and II. Available October 1982 ..... **$29.95**

TRSDOS 2.3 Decoded & Other Mysteries. James Lee Farvour. Commented Guide to TRSDOS 2.3 for the Model I. Available December 1982 ..... **$29.95**

SOFTWARE

Electric Pencil. Michael Shrayer.
Word Processing System. Available in DISK **$89.95**
STRINGY FLOPPY or CASSETTE **$79.95**

Red Pencil. Automatic Spelling Correction Program. For use with the Electric Pencil Word Processing System. Available in DISK ONLY **$89.95**

Blue Pencil. * Dictionary - Proofing Program. For use with the Electric Pencil Word Processing System. Available in DISK ONLY **$89.95**

* Requires **Red Pencil** program.

BFBLIB. Lewis Rosenfelder. Basic Faster & Better Library Disk. 32 Demonstration Programs. Basic Overlays. Video Handlers. Sorts & more for the Model I & II. Available in DISK ONLY **$19.95**

BFBDEM. Lewis Rosenfelder. Basic Faster & Better Demonstration Disk. 121 Functions, Subroutines & User Routines for the TRS-80 Model I & II. Available in DISK ONLY **$19.95**

Prices Subject to change without notice

Add $4.00 shipping and handling charge per item.
California residents add 6% sales tax. Canadian residents add 20% for exchange rate.

JG
1953 West
11th Street
Upland, CA
91786 (714)
946-5805

TRSDOS 2.3 DECODED & OTHER MYSTERIES

ISBN 0-936200-07-3