THE PROGRAMMER'S CP/M[®] HANDBOOK

Andy Johnson-Laird

Osborne/McGraw-Hill Berkeley, California Published by Osborne/McGraw-Hill 2600 Tenth Street Berkeley, California 94710 U.S.A.

For information on translations and book distributors outside of the U.S.A., please write to Osborne/McGraw-Hill at the above address.

CP/M is a registered trademark of Digital Research, Inc. CP/M-86, MP/M-86, and MP/M II are trademarks of Digital Research, Inc.

Z80 is a registered trademark of Zilog, Inc.

THE PROGRAMMER'S CP/M® HANDBOOK

Copyright ©1983 by Osborne/McGraw-Hill. All rights reserved. Printed in the United States of America. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

1234567890 DODO 89876543

ISBN 0-88134-103-7 (Paperback Edition) ISBN 0-88134-119-3 (Hardcover Edition)

Mary Borchers, Acquisitions Editor Ralph Baumgartner, Technical Editor Susan Schwartz, Copy Editor Judy Wohlfrom, Text Design Yashi Okita, Cover Design

Dedication

Several years ago I was told that "Perfection is an English education, an American salary, and a Japanese wife."

Accordingly, I wish to thank the members of Staff at Culford School in England, who gave me the English education, the people who work with me at Johnson-Laird Inc. and Control-C Software and our clients, who give me my American salary, and Mr. and Mrs. Kitagawa, who gave me Kay Kitagawa (who not only married me but took over where my English grammar left off).

A.J-L.

Acknowledgments

Although this book is not authorized or endorsed by Digital Research, I would like to express my thanks to Gary Kildall and Kathy Strutynski of Digital Research, and to Phil Nelson (formerly of Digital Research, now of Victor Technology) for their help in keeping me on the path to truth in this book. I would also like to thank Denise Penrose, Marty McNiff, Mary Borchers, and Ralph Baumgartner at Osborne/McGraw-Hill for their apparently inexhaustible patience.

A.J-L.

Contents

1	Introduction 1
2	The Structure of CP/M 5
3	The CP/M File System 17
4	The Console Command Processor (CCP)
5	The BASIC Disk Operating System 67
6	The BASIC Input/Output System 147
7	Building a New CP/M System 183
8	Writing an Enhanced BIOS 209
9	Dealing with Hardware Errors 295
10	Debugging a New CP/M System 319
11	Additional Utility Programs 371
12	Error Messages 449
Α	ASCII Character Set 465
В	CP/M Command Summary 469
С	Summary of BDOS Calls 479
D	Summary of BIOS Calls 485
	Index 487

,

45

Outline of Contents Notation Example Programs on Diskette

Introduction

This book is a sequel to the Osborne CP/M[®] User Guide by Thom Hogan. It is a technical book written mainly for programmers who require a thorough knowledge of the internal structure of CP/M — how the various pieces of CP/M work, how to use CP/M as an operating system, and finally, how to implement CP/M on different computer systems. This book is written for people who

- Have been working with microcomputers that run Digital Research's CP/M operating system.
- Understand the internals of the microprocessor world—bits, bytes, ports, RAM, ROM, and other jargon of the programmer.
- Know how to write in assembly language for the Intel 8080 or Zilog Z80 Central Processing Unit (CPU) chips.

If you don't have this kind of background, start by getting practical experience on a system running CP/M and by reading the following books from Osborne/McGraw-Hill:

• An Introduction to Microcomputers: Volume 1—Basic Concepts This book describes the fundamental concepts and facts that you need to

2 The CP/M Programmer's Handbook

know about microprocessors in order to program them. If you really need basics, there is a Volume 0 called *The Beginner's Book*.

- 8080A/8085 Assembly Language Programming This book covers all aspects of writing programs in 8080 assembly language, giving many examples.
- Osborne CP/M[®] User Guide (2nd Edition) This book introduces the CP/M operating system. It tells you how to use CP/M as a tool to get things done on a computer.

The book you are reading now deals only with CP/M Version 2.2 for the 8080 or Z80 chips. At the time of writing, new versions of CP/M and MP/M (the multi-user, multi-tasking successor to CP/M) were becoming available. CP/M-86 and MP/M-86 for the Intel 8086 CPU chip and MP/M-II for the 8080 or Z80 chips had been released, with CP/M 3.0 (8080 or Z80) in the wings. The 8086, although related architecturally to the 8080, is different enough to make it impossible to cover in detail in this book; and while MP/M-II and MP/M-86 are similar to CP/M, they have many aspects that cannot be adequately discussed within the scope of this book.

Outline of Contents

This book explains topics as if you were starting from the top of a pyramid. Successive "slices" down the pyramid cover the same material but give more detail.

The first chapter includes a brief outline of the notation used in this book for example programs written in Intel 8080 assembly language and in the C programming language.

Chapter 2 deals with the structure of CP/M, describing its major parts, their positions in memory, and their functions.

Chapter 3 discusses CP/M's file system in as much detail as possible, given its proprietary nature. The directory entry, disk parameter block, and file organization are described.

Chapter 4 covers the Console Command Processor (CCP), examining the way in which you enter command lines, the CP/M commands built into the CCP, how the CCP loads programs, and how it transfers control to these programs.

Chapter 5 begins the programming section. It deals with the system calls your programs can make to the high-level part of CP/M, the Basic Disk Operating System (BDOS).

Chapters 6 through 10 deal with the Basic Input/Output System (BIOS). This is the part of CP/M that is unique to each computer system. It is the part that you as a programmer will write and implement for your own computer system.

Chapter 6 describes a standard implementation of the BIOS.

Chapter 7 describes the mechanism for rebuilding CP/M for a different configuration.

Chapter 8 tells you how to write an enhanced BIOS.

Chapter 9 takes a close look at how to handle hardware errors—how to detect and deal with them, and how to make this task easier for the person using the computer.

Chapter 10 discusses the problems you may face when you try to debug your BIOS code. It includes debugging subroutines and describes techniques that will save you time and suffering.

Chapter 11 describes several utility programs, some that work with the features of the enhanced BIOS in Chapter 8 and some that will work with all CP/M 2 implementations.

Chapter 12 concerns error messages and some oddities that you will discover, sometimes painfully, in CP/M. Messages are explained and some probable causes for strange results are documented.

The appendixes contain "ready-reference" information and summaries of information that you need at your side when designing, coding, and testing programs to run under CP/M or your own BIOS routines.

Notation

When you program your computer, you will be sitting in front of your terminal interacting with CP/M and the utility programs that run under it. The sections that follow describe the notation used to represent the dialog that will appear on your terminal and the output that will appear on your printer.

Console Dialog

This book follows the conventions used in the Osborne CP/M User Guide, extended slightly to handle more complex dialogs. In this book

- <name> means the ASCII character named between the angle brackets,< and>. For example,<BEL> is the ASCII Bell character, and<HT> is the ASCII Horizontal Tab Character. (Refer to Appendix A for the complete ASCII character set.)
- \cdot <cr>> means to press the CARRIAGE RETURN key.
- 123 or a number without a suffix means a decimal number.
- 100B or a number followed by B means a binary number.
- 0A5H or a number followed by H means a hexadecimal number. A hexadecimal number starting with a letter is usually shown with a leading 0 to avoid confusion.

- ^x means to hold the CONTROL (CTRL) key down while pressing the x key.
- <u>Underline</u> is keyboard input you type. Output from the computer is shown without underlining.

Assembly Language Program Examples

This book uses Intel 8080 mnemonics throughout as a "lowest common denominator"—the Z80 CPU contains features absent in the 8080, but not vice versa. Output from Digital Research's ASM Assembler is shown so that you can see the generated object code as well as the source.

High-Level Language Examples

The utility programs described in Chapter 11 are written in C, a programming language which lends itself to describing algorithms clearly without becoming entangled in linguistic bureaucracy. Cryptic expressions have been avoided in favor of those that most clearly show how to solve the problem. Ample comments explain the code.

An excellent book for those who do not know how to program in C is *The C Programming Language* by Brian Kernighan and Dennis Ritchie (Prentice-Hall). Appendix A of this book is the C Reference Manual.

Example Programs on Diskette

Example programs in this book have been assembled with ASM and tested with DDT, Digital Research's Dynamic Debugging Tool. C examples were compiled using Leor Zolman's BDS C Compiler (Version 1.50) and tested using the enhanced BIOS described in Chapter 8.

All of the source code shown in this book is available on a single-sided, single-density, 8-inch diskette (IBM 3740 format). Please do *not* contact Osborne/ McGraw-Hill to order this diskette. Call or write

> Johnson-Laird, Inc. Attn: The CP/M Programmer's Handbook Diskette 6441 SW Canyon Court Portland, OR 97221 Tel: (503) 292-6330

The diskette is available for \$50 plus shipping costs.

CP/M from Digital Research The Pieces of CP/M CP/M Diskette Format Loading CP/M Console Command Processor Basic Disk Operating System Basic Input/Output System CCP, BDOS, and BIOS Interactions



The Structure of CP/M

This chapter introduces the pieces that make up CP/M — what they are and what they do. This bird's-eye view of CP/M will establish a framework to which later chapters will add more detailed information.

You may have purchased the standard version of CP/M directly from Digital Research, but it is more likely you received CP/M when you bought your microprocessor system or its disk drive system. Or, you may have purchased CP/M separately from a software distributor. In any case, this distributor or the company that made the system or disk drive will have already modified the standard version of CP/M to work on your specific hardware. Most manufacturers' versions of CP/M have more files on their system diskette than are described here for the standard Digital Research release.

Some manufacturers have rewritten all the documentation so that you may not have received any Digital Research CP/M manuals. If this is the case, you should order the complete set from Digital Research, because as a programmer, you will need to have them for reference.

CP/M from Digital Research

Digital Research provides a standard "vanilla-flavored" version of CP/M that will run only on the Intel Microcomputer Development System (MDS). The CP/M package from Digital Research contains seven manuals and an 8-inch, single-sided, single-density standard IBM 3740 format diskette.

The following manuals come with this CP/M system:

- An Introduction to CP/M Features and Facilities. This is a brief description of CP/M and the utility programs you will find on the diskette. It describes only CP/M version 1.4.
- *CP/M 2.0 User's Guide*. Digital Research wrote this manual to describe the new features of CP/M 2.0 and the extensions made to existing CP/M 1.4 features.
- ED: A Context Editor for the CP/M Disk System. By today's standards, ED is a primitive line editor, but you can still use it to make changes to files containing ASCII text, such as the BIOS source code.
- *CP/M Assembler (ASM).* ASM is a simple but fast assembler that can be used to translate the BIOS source code on the diskette into machine code. Since ASM is only a bare-bones assembler, many programmers now use its successor, MAC (also from Digital Research).
- *CP/M Dynamic Debugging Tool (DDT).* DDT is an extremely useful program that allows you to load programs in machine code form and then test them, executing the program either one machine instruction at a time or stopping only when the CPU reaches a specific point in the program.
- *CP/M Alteration Guide*. There are two manuals with this title, one for CP/M version 1.4 and the other for 2.0. Both manuals describe, somewhat cryptically, how to modify CP/M.
- *CP/M Interface Guide*. Again, there are two versions, 1.4 and 2.0. These manuals tell you how to write programs that communicate directly with CP/M.

The diskette supplied by Digital Research has the following files:

ASM.COM

The CP/M assembler.

BIOS.ASM

A source code file containing a sample BIOS for the Intel Microcomputer Development System (MDS). Unless you have the MDS, this file is useful only as an example of a BIOS.

CBIOS.ASM

Another source code file for a BIOS. This one is skeletal: There are gaps so that you can insert code for your computer.

DDT.COM

The Dynamic Debugging Tool program.

DEBLOCK.ASM

A source code file that you will need to use in the BIOS if your computer uses sector sizes other than 128 bytes. It is an example of how to block and deblock 128-byte sectors to and from the sector size you need.

DISKDEF.LIB

A library of source text that you will use if you have a copy of Digital Research's advanced assembler, MAC.

DUMP.ASM

The source for an example program. DUMP reads a CP/M disk file and displays it in hexadecimal form on the console.

DUMP.COM

The actual executable program derived from DUMP.ASM.

ED.COM

The source file editor.

LOAD.COM

A program that takes the machine code file output by the assembler, ASM, and creates another file with the data rearranged so that you can execute the program by just typing its name on the keyboard.

MOVCPM.COM

A program that creates versions of CP/M for different memory sizes.

PIP.COM

A program for copying information from one place to another (PIP is short for Peripheral Interchange Program).

STAT.COM

A program that displays statistics about the CP/M and other information that you have stored on disks.

SUBMIT.COM

A program that you use to enter CP/M commands automatically. It helps you avoid repeated typing of long command sequences.

SYSGEN.COM

A program that writes CP/M onto diskettes.

XSUB.COM

An extended version of the SUBMIT program. The files named previously

fall into two groups: One group is used only to rebuild CP/M, while the other set is general-purpose programming tools.

The Pieces of CP/M

CP/M is composed of the Basic Disk Operating System (BDOS), the Console Command Processor (CCP), and the Basic Input/Output System (BIOS).

On occasion you will see references in CP/M manuals to something called the FDOS, which stands for "Floppy Disk Operating System." This name is given to the portion of CP/M consisting of both the BDOS and BIOS and is a relic passed down from the original version. Since it is rarely necessary to refer to the BDOS and the BIOS combined as a single entity, no further references to the FDOS will be made in this book.

The BDOS and the CCP are the proprietary parts of CP/M. Unless you are willing to pay several thousand dollars, you cannot get the source code for them. You do not need to. CP/M is designed so that all of the code that varies from one machine to another is contained in the BIOS, and you do get the BIOS source code from Digital Research. Several companies make specialized BIOSs for different computer systems. In many cases they, as well as some CP/M hardware manufacturers, do not make the source code for their BIOS available; they have put time and effort into building their BIOS, and they wish to preserve the proprietary nature of what they have done.

You may have to build a special configuration of CP/M for a specific computer. This involves no more than the following four steps:

- 1. Make a version of the BDOS and CCP for the memory size of your computer.
- 2. Write a modified version of the BIOS that matches the hardware in your computer.
- 3. Write a small program to load CP/M into memory when you press the RESET button on your computer.
- 4. Join all of the pieces together and write them out to a diskette.

These steps will be explained in Chapters 7, 8, and 9.

In the third step, you write a small program that loads CP/M into memory when you press the RESET button on your computer. This program is normally called the bootstrap loader. You may also see it called the "boot" or even the "cold start" loader. "Bootstrap" refers to the idea that when the computer is first turned on, there is no program to execute. The task of getting that very first program into the computer is, conceptually, as difficult as attempting to pick yourself up off the ground by pulling on your own bootstraps. In the early days of computing, this operation was performed by entering instructions manually—setting large banks of switches (the computer was built to read the switches as soon as it was turned on). Today, microcomputers contain some small fragment of a program in "nonvolatile" read-only memory (ROM) — memory that retains data when the computer is turned off. This stored program, usually a Programmable Read Only Memory (PROM) chip, can load your bootstrap program, which in turn loads CP/M.

CP/M Diskette Format

The standard version of CP/M is formatted on an 8-inch, single-sided diskette. Diskettes other than this type will probably have different layouts; hard disks definitely will be different.

The physical format of the standard 8-inch diskette is shown in Figure 2-1. The

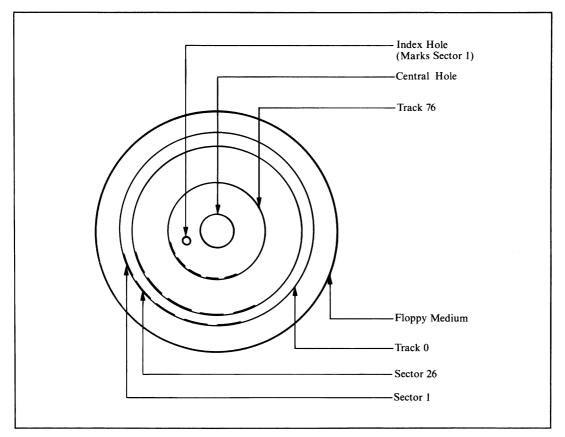


Figure 2-1. Floppy disk layout

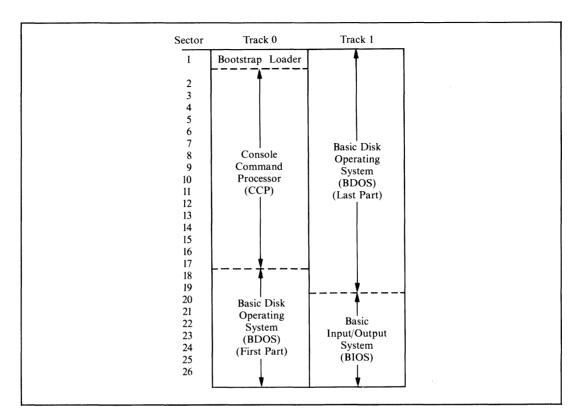


Figure 2-2. Layout of CP/M on tracks 0 and 1 of floppy disk

diskette has a total of 77 concentric tracks numbered from zero (the outermost) to 76 (the innermost). Each of these tracks is divided radially into 26 sectors. These physical sectors are numbered from 1 to 26; physical sector zero does not exist. Each sector has enough space for 128 bytes of data.

Even when CP/M is implemented on a large hard disk with much larger sector sizes, it still works with 128-byte sectors. The BIOS has extra instructions that convert the *real* sectors into CP/M-style 128-byte sectors.

A final note on physical format: The soft-sectored, single-sided, single-density, 8-inch diskette (IBM 3740 format) is the *only* standard format. Any other formats will be unique to the hardware manufacturer that uses them. It is unlikely that you can read a diskette on one manufacturer's computer if it was written on another's, even though the formats appear to be the same. For example, a single-sided, double-density diskette written on an Intel Development System cannot be read on a Digital Microsystems computer even though both use double-density format. If you want to move data from one computer to another, use 8-inch, single-sided, single-density format diskettes, and it *should* work. In order to see how CP/M is stored on a diskette, consider the first two tracks on the diskette, track 0 and track 1. Figure 2-2 shows how the data is stored on these tracks.

Loading CP/M

The events that occur after you first switch on your computer and put the CP/M diskette into a disk drive are the same as those that occur when you press the RESET button—the computer generates a RESET signal.

The RESET button stops the central processor unit (CPU). All of the internals of the CPU are set to an initial state, and all the registers are cleared to zero. The program counter is also cleared to zero so that when the RESET signal goes away (it only lasts for a few milliseconds), the CPU starts executing instructions at location 0000H in memory.

Memory chips, when they first receive power, cannot be relied upon to contain any particular value. Therefore, hardware designers arrange for some initial instructions to be forced into memory at location 0000H and onward. It is this feat that is like pulling yourself up by your own bootstraps. How can you make the computer obey a particular instruction when there is "nothing" (of any sensible value) inside the machine?

There are two common techniques for placing preliminary instructions into memory:

Force-feeding

With this approach, the hardware engineer assumes that when the RESET signal is applied, some part of the computer system, typically the floppy disk controller, can masquerade as memory. Just before the CPU is unleashed, the floppy disk controller will take control of the computer system and copy a small program into memory at location 0000H and upward. Then the CPU is allowed to start executing instructions at location 0000H. The disk controller preserves the instructions even when power is off because they are stored in nonvolatile PROM-based firmware. These instructions make the disk controller read the first sector of the first track of the system diskette into memory and then transfer control to it.

Shadow ROM

This is a variation of the force-feeding technique. The hardware manufacturer arranges some ROM at location 0000H. There is also some normal read/write memory at location 0000H, but this is electronically disabled when the RESET signal has been activated. The CPU, unleashed at location 0000H, starts to execute the ROM instruction. The first act of the ROM program is to copy itself into read/write memory at some convenient location higher up in memory and transfer control of the machine up to this copy. Then the real memory at location 0000H can be turned on, the ROM turned off, and the first sector on the disk read in. With either technique, the result is the same. The first sector of the disk is read into memory and control is transferred to the first instruction contained in the sector.

This first sector contains the main CP/M bootstrap program. This program initializes some aspects of the hardware and then reads in the remainder of track 0 and most of the sectors on track 1 (the exact number depends on the overall length of the BIOS itself). The CP/M bootstrap program will contain only the most primitive diskette error handling, trying to read the disk over and over again if the hardware indicates that it is having problems reading a sector.

The bootstrap program loads CP/M to the correct place in memory; the load address is a constant in the bootstrap. If you need to build a version of CP/M that uses more memory, you will need to change this load address inside the bootstrap as well as the address to which the bootstrap will jump when all of CP/M has been read in. This address too is a constant in the bootstrap program.

The bootstrap program transfers control to the first instruction in the BIOS, the cold boot entry point. "Cold" implies that the operation is starting cold from an empty computer.

The cold boot code in the BIOS will set up the hardware in your computer. That is, it programs the various chips that control the speed at which serial ports transmit and receive data. It initializes the serial port chips themselves and generally readies the computer system. Its final act is to transfer control to the first instruction in the BDOS in order to start up CP/M proper.

Once the BDOS receives control, it initializes itself, scans the file directory on the system diskette, and hands over control to the CCP. The CCP then outputs the "A>" prompt to the console and waits for you to enter a command. CP/M is then ready to do your bidding.

At this point, it is worthwhile to review which CP/M parts are in memory, where in memory they are, and what functions they perform.

This overview will look at memory first. Figure 2-3 shows the positions in memory of the Console Command Processor, the Basic Disk Operating System, and the Basic Input/Output System.

By touching upon these major memory components—the CCP, BDOS, and BIOS—this discussion will consider which modules interact with them, how requests for action are passed to them, and what functions they can perform.

Console Command Processor

As you can see in Figure 2-3, the CCP is the first part of CP/M that is encountered going "up" through memory addresses. This is significant when you consider that the CCP is only necessary in between programs. When CP/M is idle, it needs the CCP to interact with you, to accept your next command. Once CP/M has started to execute the command, the CCP is redundant; any console interaction will be handled by the program you are running rather than by the CCP.

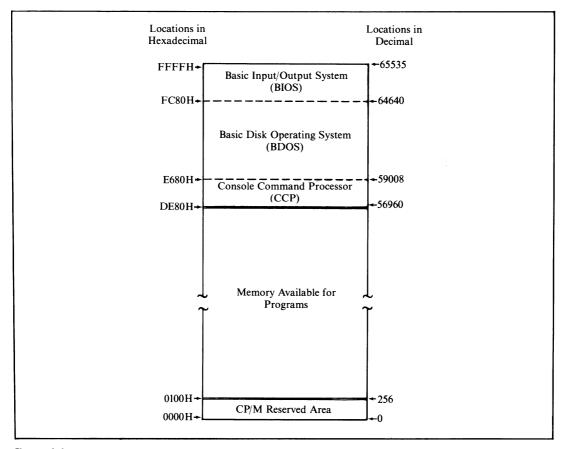


Figure 2-3. Memory layout with CP/M loaded

Therefore, the CCP leads a very jerky existence in memory. It is loaded when you first start CP/M. When you ask CP/M, via the CCP, to execute a program, this program can overwrite the CCP and use the memory occupied by the CCP for its own purposes. When the program you asked for has finished, CP/M needs to reload the CCP, now ready for its interaction with you. This process of reloading the CCP is known as a *warm boot*. In contrast with the cold boot mentioned before, the warm boot is not a complete "start from cold"; it's just a reloading of the CCP. The BDOS and BIOS are not touched.

How does a program tell CP/M that it has finished and that a warm boot must be executed? By jumping to location 0000 H. While the BIOS was initializing itself during the cold boot routine, it put an instruction at location 0000 H to jump to the warm boot routine, which is also in the BIOS. Once the BIOS warm boot routine

14 The CP/M Programmer's Handbook

has reloaded the CCP from the disk, it will transfer control to the CCP. (The cold and warm boot routines are discussed further in Chapter 6.)

This brief description indicates that every command you enter causes a program to be loaded, the CCP to be overwritten, the program to run, and the CCP to be reloaded when the program jumps to location 0000H on completing its task. This is not completely true. Some frequently needed commands reside in the CCP. Using one of these commands means that CP/M does not have to load anything from a diskette; the programs are already in memory as part of the CCP. These commands, known as "intrinsic" or "resident" commands, are listed here with a brief description of what they do. (All of them are described more thoroughly in Chapter 4.) The "resident" commands are

DIR	Displays which files are on a diskette
ERA	Erases files from a diskette
REN	Changes the names of files on diskette
TYPE	Displays the contents of text files on the console
SAVE	Saves some of memory as a file on diskette
USER	Changes User File Group.

Basic Disk Operating System

The BDOS is the heart of CP/M. The CCP and all of the programs that you run under CP/M talk to the BDOS for all their outside contacts. The BDOS performs such tasks as console input/ output, printer output, and file management (creating, deleting, and renaming files and reading and writing sectors).

The BDOS performs all of these things in a rather detached way. It is concerned only with the logical tasks at hand rather than the detailed action of getting a sector from a diskette into memory, for example. These "low-level" operations are done by the BDOS in conjunction with the BIOS.

But how does a program work with the BDOS? By another strategically placed jump instruction in memory. Remember that the cold boot placed the jump to the BIOS warm boot routine in location 0000H. At location 0005H, it puts a jump instruction that transfers control up to the first instruction of the BDOS. Thus, any program that transfers control to location 0005H will find its way into the BDOS. Typically, programs make a CALL instruction to location 0005H so that once the BDOS has performed the task at hand, it can return to the calling program at the correct place. The program enlisting the BDOS's help puts special values into several of the CPU registers before it makes the call to location 0005H. These values tell the BDOS what operation is required and the other values needed for the specific operation.

Basic Input/Output System

As mentioned before, the BDOS deals with the input and output of information in a detached way, unencumbered by the physical details of the computer hardware. It is the BIOS that communicates directly with the hardware, the ports, and the peripheral devices wired to them.

This separation of *logical* input/output in the BDOS from the *physical* input/ output in the BIOS is one of the major reasons why CP/M is so popular. It means that the same version of CP/M can be adapted for all types of computers, regardless of the oddities of the hardware design. Digital Research will tell you that there are over 200,000 computers in the world running CP/M. Just about all of them are running *identical* copies of the CCP and BDOS. Only the BIOS is different. If you write a program that plays by the rules and only interacts with the BDOS to get things done, it will run on almost all of those 200,000 computers without your having to change a single line of code.

You probably noticed the word "almost" in the last paragraph. Sometimes programmers make demands of the BIOS directly rather than the BDOS. This leads to trouble. The BIOS should be off limits to your program. You need to know what it is and how it works in order to build a customized version of CP/M, but you must *never* write programs that talk directly to the BIOS if you want them to run on other versions of CP/M.

Now that you understand the perils of talking to the BIOS, it is safe to describe how the BDOS communicates with the BIOS. Unlike the BDOS, which has a single entry point and uses a value in a register to specify the function to be performed, the BIOS has several entry points. The first few instructions in the BIOS are all independent entry points, each taking up three bytes of memory. The BDOS will enter the BIOS at the appropriate instruction, depending on the function to be performed. This group of entry points is similar in function to a railroad marshalling yard. It directs the BDOS to the correct destination in the BIOS for the function it needs to have done. The entry point group consists of a series of JUMP instructions, each one three bytes long. The group as a whole is called the BIOS jump table, or jump vector. Each entry point has a predefined meaning. These points are detailed and will be discussed in Chapter 6.

CCP, BDOS, and BIOS Interactions

Figure 2-4 summarizes the functions that the CCP, BDOS, and BIOS perform, the ways in which these parts of CP/M communicate among themselves, and the way in which one of your programs running under CP/M interacts with the BDOS.

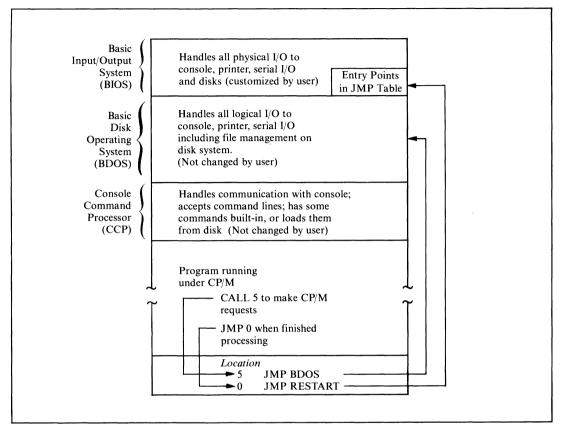


Figure 2-4. CP/M's functional breakdown

How CP/M Views the Disk The Making of a File Disk Definition Tables File Organizations



The CP/M File System

This chapter gives you a close look at the CP/M file system. The Basic Disk Operating System (BDOS) is responsible for this file system: It keeps a directory of the files on disk, noting where data are actually stored on the disk. Because the file system automatically keeps track of this information, you can ignore the details of which tracks and sectors on the disk have data for a given file.

How CP/M Views the Disk

To manage files on the disk, CP/M works with the disk in logical terms rather than in physical terms of tracks and sectors. CP/M treats the disk as three major areas.

These are the *reserved area*, which contains the bootstrap program and CP/M itself; the *file directory*, containing one or more entries for each file stored on the disk; and the *data storage area*, which occupies the remainder of the disk. You will

be looking at how CP/M allocates the storage to the files as your programs create them.

The Basic Input/Output System (BIOS) has built-in tables that tell CP/M the respective sizes of the three areas. These are the *disk definition tables*, described later in this chapter.

Allocation Blocks

Rather than work with individual 128-byte sectors, CP/M joins several of these sectors logically to form an allocation block. Typically, an allocation block will contain eight 128-byte sectors (which makes it 1024 or 1K bytes long). This makes for easier disk manipulation because the magnitude of the numbers involved is reduced. For example, a standard 8-inch, single-density, single-sided floppy disk has 1950 128-byte sectors; hard disks may have 120,000 or more. By using allocation blocks that view the disk eight sectors at a time, the number of storage units to be managed is substantially reduced. The total number is important because numeric information is handled as 16-bit integers on the 8080 and Z80 microprocessors, and therefore the largest unsigned number possible is 0FFFFH (65,535 or 64K decimal).

Whenever CP/M refers to a specific allocation block, all that is needed is a simple number. The first allocation block is number 0, the next is number 1, and so on, up to the total remaining capacity of the disk.

The typical allocation block contains 1024 (1K) bytes, or eight 128-byte sectors. For the larger hard disks, the allocation block can be 16,384 (16K) bytes, which is 128 128-byte sectors. CP/M is given the allocation via an entry in the disk definition tables in the BIOS.

The size of the allocation block is not arbitrary, but it is a compromise. The originator of the working BIOS for the system—either the manufacturer or the operating system's designer—chooses the size by considering the total storage capacity of the disk. This choice is tempered by the fact that if a file is created with only a single byte of data in it, that file would be given a complete allocation block. Large allocation blocks can waste disk storage if there are many small files, but they can be useful when a few very large files are called for.

This can be seen better by considering the case of a 1K-byte allocation block. If you create a very small file containing just a single byte of data, you will have allocated an entire allocation block. The remaining 1023 bytes will not be used. You can use them by adding to the file, but when you first create this one-byte file, they will be just so much dead space. This is the problem: Each file on the disk will normally have one partly filled allocation block. If these blocks are very large, the amount of wasted (unused) space can be very large. With 16K-byte blocks, a 10-megabyte disk with only 3 megabytes of data on it could become logically full, with all allocation blocks allocated.

On the other hand, when you use large allocation blocks, CP/M's performance is significantly improved because the BDOS refers to the file directory less frequently. For example, it can read a 16K-byte file with only a single directory reference.

Therefore, when considering block allocation, keep the following questions in mind:

How big is the logical disk?

With a larger disk, you can tolerate space wasted by incomplete allocation blocks.

What is the mean file size?

If you anticipate many small files, use small allocation blocks so that you have a larger "supply" of blocks. If you anticipate a smaller number of large files, use larger allocation blocks to get faster file operations.

When a file is first created, it is assigned a single allocation block on the disk. Which block is assigned depends on what other files you already have on the disk and which blocks have already been allocated to them. CP/M maintains a table of which blocks are allocated and which are available. As the file accumulates more data, it will fill up the first allocation block. When this happens, CP/M will extend the file and allocate another block to it. Thus, as the file grows, it occupies more blocks. These blocks need not be adjacent to each other on the disk. The file can exist as a series of allocation blocks scattered all over the disk. However, when you need to see the entire file, CP/M presents the allocation blocks in the correct order. Thus, application programs can ignore allocation blocks. CP/M keeps track of which allocation blocks belong to each file through the file directory.

The File Directory

The *file directory* is sandwiched between the reserved area and the data storage area on the disk. The actual size of the directory is defined in the BIOS's disk definition tables. The directory can have some binary multiple of entries in it, with one or more entries for each file that exists on the disk. For a standard 8-inch floppy diskette, there will be room for 64 directory entries; for a hard disk, 1024 entries would not be unusual. Each directory entry is 32 bytes long.

Simple arithmetic can be used to calculate how much space the directory occupies on a standard floppy diskette. For example, for a floppy disk the formula is $64 \times 32 = 2048$ bytes = 2 allocation blocks of 1024 bytes each.

The directory entry contains the name of the file along with a list of the allocation blocks currently used by the file. Clearly, a single 32-byte directory entry cannot contain all of the allocation blocks necessary for a 5-megabyte file, especially since CP/M uses only 16 bytes of the 32-byte total for storage of allocation block numbers.

Extents

Often CP/M will need to control files that need many allocation blocks. It does this by creating more than one directory entry. Second and subsequent directory

entries have the same file name as the first. One of the other bytes of the directory entry is used to indicate the directory entry sequence number. Each new directory entry brings with it a new supply of bytes that can be used to hold more allocation block numbers. In CP/M jargon, each directory entry is called an *extent*. Because the directory entry for each extent has 16 bytes for storing allocation block numbers, it can store either 16 one-byte numbers or 8 two-byte numbers. Therefore, the total number of allocation blocks possible in each extent is either 8 (for disks with more than 255 allocation blocks) or 16 (for smaller disks).

File Control Blocks

Before CP/M can do anything with a file, it has to have some control information in memory. This information is stored in a *file control block*, or FCB. The FCB has been described as a motel for directory entries—a place for them to reside when they are not at home on the disk. When operations on a file are complete, CP/M transforms the FCB back into a directory entry and rewrites it over the original entry. The FCB is discussed in detail at the end of this chapter.

As a summary, Figure 3-1 shows the relationships between disk sectors, allocation blocks, directory entries, and file control blocks.

The Making of a File

To reinforce what you already know about the CP/M file system, this section takes you on a "walk-through" of the events that occur when a program running under CP/M creates a file, writes data to it, and then *closes* the file.

Assume that a program has been loaded in memory and the CPU is about to start executing it. First, the program will declare space in memory for an FCB and will place some preset values there, the most important of which is the file name. The area in the FCB that will hold the allocation block numbers as they are assigned is initially filled with binary 0's. Because the first allocation block that is available for file data is block 1, an allocation block number of 0 will mean that no blocks have been allocated.

The program starts executing. It makes a call to the BDOS (via location 0005H) requesting that CP/M create a file. It transfers to the BDOS the address in memory of the FCB. The BDOS then locates an available entry in the directory, creates a new entry based on the FCB in the program, and returns to the program, ready to write data to the file. Note that CP/M makes no attempt to see if there is already a file of the same name on the disk. Therefore, most real-world programs precede a request to make a file with a request to delete any existing file of the same name.

The program now starts writing data to the file, 128-byte sector by 128-byte sector. CP/M does not have any provision for writing one byte at a time. It handles data sector-by-sector only, flushing sectors to the disk as they become full.

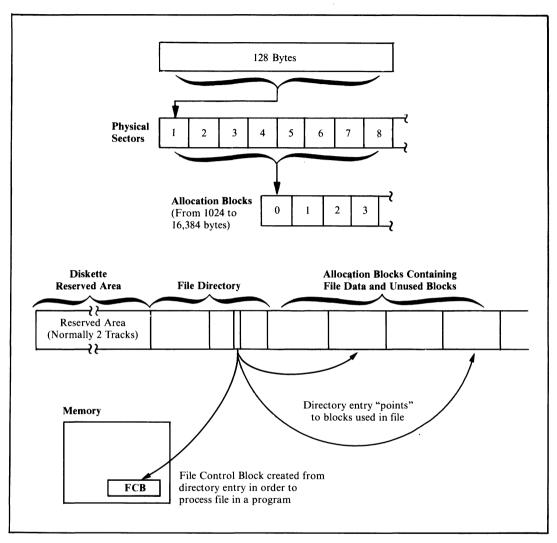


Figure 3-1. The hierarchical relationship between sectors, allocation blocks, directory entires, and FCBs

The first time a program asks CP/M (via a BDOS request) to write a sector onto the file on the disk, the BDOS finds an unused allocation block and assigns it to the file. The number of the allocation block is placed inside the FCB in memory. As each allocation block is filled up, a new allocation block is found and assigned, and its number is added to the list of allocation blocks inside the FCB. Finally, when the FCB has no more room for allocation block numbers, the BDOS

• Writes an updated directory entry out to the disk.

- Seeks out the next spare entry in the directory.
- Resets the FCB in memory to indicate that it is now working on the second extent of the file.
- Clears out the allocation block area in the FCB and waits for the next sector from the program.

Thus the process continues. New extents are automatically opened until the program determines that it is time to finish, writes the last sector out to the disk, and makes a BDOS request to close the file. The BDOS then converts the FCB into a final directory entry and writes to the directory.

Directory Entry

The directory consists of a series of 32-byte entries with one or more entries for each file on the disk. The total number of entries is a binary multiple. The actual number depends on the disk format (it will be 64 for a standard floppy disk and perhaps 2048 for a hard disk).

Figure 3-2 shows the detailed structure of a directory entry. Note that the description is actually Intel 8080 source code for the data definitions you would need in order to manipulate a directory entry. It shows a series of EQU instructions—*equate* instructions, used to assign values or expressions to a label, and in this case used to access an entry. It also shows a series of DS or *define storage* instructions used to declare storage for an entry. The comments on each line describe the function of each of the fields. Where data elements are less than a byte long, the comment identifies which bits are used.

As you study Figure 3-2, you will notice some terminology that as yet has not been discussed. This is described in detail in the sections that follow.

File User Number (Byte 0) The least significant (low order) four bits of byte 0 in the directory entry contain a number in the range 0 to 15. This is the *user number* in which the file belongs. A better name for this field would have been file group number. It works like this: Suppose several users are sharing a computer system with a hard disk that cannot be removed from the system without a lot of trouble. How can each user be sure not to tamper with other users' files? One simple way would be for each to use individual initials as the first characters of any file names. Then each could tell at a glance whether a file was another's and avoid doing anything to anyone else's files. A drawback of this scheme is that valuable character positions would be used in the file name, not to mention the problems resulting if several users had the same initials.

The file user number is prefixed to each file name and can be thought of as part of the name itself. When CP/M is first brought up, User 0 is the default user—the one that will be chosen unless another is designated. Any files created will go into the directory bearing the user number of 0. These files are referred to as being in user area 0. However, with a shared computer system, arrangements must be made for multiple user areas. The USER command makes this possible. User numbers and areas can range from 0 through 15. For example, a user in area 7 would not be able to get a directory of, access, or erase files in user area 5.

This user-number byte serves a second purpose. If this byte is set to a value of 0E5H, CP/M considers that the file directory entry has been deleted and completely ignores the remaining 31 bytes of data. The number 0E5H was not chosen whimsically. When IBM first defined the standard for floppy diskettes, they chose the binary pattern 11100101 (0E5H) as a good test pattern. A new floppy diskette formatted for use has nothing but bytes of 0E5H on it. Thus, the process of erasing a file is a "logical" deletion, where only the first byte of the directory entry is changed to 0E5H. If you accidentally delete a file (and provided that no other directory activity has occurred) it can be resurrected by simply changing this first byte back to a reasonable user number. This process will be explained in Chapter 11.

File Name and Type (Bytes 1 - 8 and 9 - 11) As you can see from Figure 3-2, the file name in a directory entry is eight bytes long; the file type is three. These two fields are used to name a file unambiguously. A file name can be less than eight characters and the file type less than three, but in these cases, the unused character positions are filled with spaces.

Whenever file names and file types are written together, they are separated by a period. You do not need the period if you are not using the file type (which is the same as saying that the file type is all spaces). Some examples of file names are

READ. ME LONGNAME.TYP 1 1.2

0000		FDE\$USER	EQU	0	;File user number (LS 4 bits)
0001	=	FDE\$NAME	EQU	1	;file name (8 bytes)
0005) =	FDE\$TYP	EQU	9	;File type
					;Offsets for bits used in type
0005		FDE\$R0	EQU	9	;Bit 7 = 1 - Read only
000A	-	FDE\$SYS	EQU	10	;Bit 7 = 1 - System status
000E	=	FDE\$CHANGE	EQU	11	;Bit 7 = 0 = File Written To
					;
0000	=	FDE\$EXTENT	EQU	12	Extent number
					:13, 14 reserved for CP/M
000F	=	FDE\$RECUSED	EQU	15	Records used in this extent;
0010	=	FDE\$ABUSED	EQU	16	Allocation blocks used
		;			
		;			
		;			
0000		FD\$USER:	DS		;File user number
0001		FD\$NAME:	DS	8	;File name
0009		FD\$TYP:	DS	3	;File type
0000		FD\$EXTENT:	DS	1	;Extent
0000		FD\$RESV:	DS	2	;Reserved for CP/M
000F		FD\$RECUSED:	DS	1	;Records used in this extent
0010		FD\$ABUSED:	DS	16	Allocation blocks used

Figure 3-2. Data declarations for CP/M's file directory entries

A file name and type can contain the characters A through Z, 0 through 9, and some of the so-called "mark" characters such as "/" and "—". You can also use lowercase letters, but be careful. When you enter commands into the system using the CCP, it converts all lowercases to uppercases, so it will never be able to find files that actually have lowercase letters in their directory entries. Avoid using the "mark" characters excessively. Ones you can use are

!@#\$%()-+/

Characters that you must not use are

<>.,;:=?*[]

These characters are used by CP/M in normal command lines, so using them in file names will cause problems.

You can use odd characters in file names to your advantage. For example, if you create files with nongraphic characters in their names or types, the only way you can access these files will be from within programs. You cannot manipulate these files from the keyboard except by using ambiguous file names (described in the next section). This makes it more difficult to erase files accidentally since you cannot specify their names directly from the console.

Ambiguous File Names CP/M has the capability to refer to one or more file names by using special "wild card" characters in the file names. The "?" is the main wildcard character. Whenever you ask CP/M to do something related to files, it will match a "?" with any character it finds in the file name. In the extreme case, a file name and type of "?????????" will match with any and all file names.

As another example, all the chapters of this book were held in files called "CHAP1.DOC," "CHAP2.DOC," and so on. They were frequently referred to, however, as "CHAP??.DOC." Why two question marks? If only one had been used, for example, "CHAP?.DOC," CP/M would not have been able to match this with "CHAP10.DOC" nor any other chapter with two digits. The matching that CP/M does is strictly character-by-character.

Because typing question marks can be tedious and special attention must be paid to the exact number entered, a convenient shorthand is available. The asterisk character "*" can be used to mean "as many ?'s as you need to fill out the name or the type field." Thus, "?????????" can be written "*.*" and "CHAP??.DOC" could also be rewritten "CHAP*.DOC."

The use of "*" is allowed only when you are entering file names from the console. The question mark notation, however, can be used for certain BDOS operations, with the file name and type field in the FCB being set to the "?" as needed.

File Type Conventions Although you are at liberty to think up file names without constraint, file types are subject to convention and, in one or two cases, to the mandate of CP/M itself.

The types that will cause problems if you do not use them correctly are

.ASM

Assembly language source for the ASM program

.MAC

Macro assembly language

.HEX

Hexadecimal file output by assemblers

.REL

Relocatable file output by assemblers

.COM

Command file executed by entering its name alone

.PRN

Print file written to disk as a convenience

.LIB

Library file of programs

.SUB

Input for CP/M SUBMIT utility program

Examples of conventional file types are

.*C*

C source code

.PAS

Pascal source code

.COB

COBOL source code

.FTN

FORTRAN source code

.APL

APL programs

.TXT

Text files

.DOC

Documentation files

.INT

Intermediate files

.DTA

Data files

.IDX Index files .\$\$\$ Temporary files

The file type is also useful for keeping several copies of the same file, for example, "TEST.001," "TEST.002," and so on.

File Status Each one of the states *Read-Only, System*, and *File Changed* requires only a single bit in the directory entry. To avoid using unnecessary space, they have been slotted into the three bytes used for the file type field. Since these bytes are stored as characters in ASCII (which is a seven-bit code), the most significant bit is not used for the file type and thus is available to show status.

Bit 7 of byte 9 shows Read-Only status. As its name implies, if a file is set to be Read-Only, CP/M will not allow any data to be written to the file or the file to be deleted.

If a file is declared to be System status (bit 7 of byte 10), it will not show up when you display the file directory. Nor can the file be copied from one place to another with standard CP/M utilities such as PIP unless you specifically ask the utility to do so. In normal practice, you should set your standard software tools and application programs to be both Read-Only and System status/ Read-Only, so that you cannot accidentally delete them, and System status, so that they do not clutter up the directory display.

The File Changed bit (bit 7 of byte 11) is always set to 0 when you close a file to which you have been writing. This can be useful in conjunction with a file backup utility program that sets this bit to 1 whenever it makes a backup copy. Just by scanning the directory, this utility program can determine which files have changed since it was last run. The utility can be made to back up only those files that have changed. This is much easier than having to remember which files you have changed since you last made backup copies.

With a floppy disk system, there is less need to worry about backing up on a file-by-file basis — it is just as easy to copy the whole diskette. This system is useful, however, with a hard disk system with hundreds of files stored on the disk.

File Extent (Byte 12) Each directory entry represents a file extent. Byte 12 in the directory entry identified the extent number. If you have a file of less than 16,384 bytes, you will need only one extent—number 0. If you write more information to thie file, more extents will be needed. The extent number increases by 1 as each new extent is created.

The extent number is stored in the file directory because the directory entries are in random sequence. The BDOS must do a sequential search from the top of the directory to be sure of finding any given extent of a file. If the directory is large, as it could be on a hard disk system, this search can take several seconds. **Reserved Bytes 13 and 14** These bytes are used by the proprietary parts of CP/M's file system. From your point of view, they will be set to 0.

Record Number (Byte 15) Byte 15 contains a count of the number of records (128-byte sectors) that have been used in the last partially filled allocation block referenced in this directory entry. Since CP/M creates a file sequentially, only the most recently allocated block is not completely full.

Disk Map (Bytes 16-31) Bytes 16-31 store the allocation block numbers used by each extent. There are 16 bytes in this area. If the total number of allocation blocks (as defined by you in the BIOS disk tables) is less than 256, this area can hold as many as 16 allocation block numbers. If you have described the disk as having more than 255 allocation blocks, CP/M uses this area to store eight two-byte values. In this case allocation blocks can take on much larger values.

A directory entry can store either 8 or 16 allocation block numbers. If the file has not yet expanded to require this total number of allocation blocks, the unused positions in the entry are filled with zeros. You may think this would create a problem because it appears that several files will have been allocated block 0 over and over. In fact, there is no problem because the file directory itself always occupies block 0 (and depending on its size several of the blocks following). For all practical purposes, block 0 "does not exist," at least for the storage of file data.

Note that if, by accident, the relationship between files and their allocation blocks is scrambled—that is, either the data in a given block is overwritten, or two or more active directory entries contain the same block number—CP/M cannot access information properly and the disk becomes worthless.

Several commercially available utility programs manipulate the directory. You can use them to inspect and change a damaged directory, reviving accidentally erased files if you need to. There are other utilities you can use to logically remove bad sectors on the disk. These utilities find the bad areas, work backward from the track and sector numbers, and compute the allocation block in which the error occurs. Once the block numbers are known, they create a dummy file, either in user area 15 or, in some cases, in an "impossible" user area (one greater than 15), that appears to "own" all the bad allocation blocks.

A good utility program protects the integrity of the directory by verifying that each allocation block is "owned" by only one directory entry.

Disk Definition Tables

As mentioned previously, the BIOS contains tables telling the BDOS how to view the disk storage devices that are part of the computer system. These tables are built by you. If you are using standard 8-inch, single-sided, single-density floppy

diskettes, you can use the examples in the Digital Research manual CP/M 2Alteration Guide. But if you are using some other, more complex system, you must make some careful judgments. Any mistakes in the disk definition tables can create serious problems, especially when you try to correct diskettes created using the erroneous tables. You, as a programmer, must ensure the correctness of the tables by being careful.

One other point before looking at table structures: Because the tables exist and define a particular disk "shape" does not mean that such a disk need necessarily be connected to the system. The tables describe *logical* disks, and there is no way for the physical hardware to check whether your disk tables are correct. You may have a computer system with a single hard disk, yet describe the disk as though it were divided into several *logical* disks. CP/M will view each such "disk" independently, and they should be thought of as separate disks.

Disk Parameter Header Table

This table is the starting point in the disk definition tables. It is the topmost structure and contains nothing but the addresses of other structures. There is one entry in this table for each logical disk that you choose to describe. There is an entry point in the BIOS that returns the address of the parameter header table for a specific logical disk.

An example of the code needed to define a disk parameter header table is shown in Figure 3-3.

Sector Skewing (Skewtable) To define sector *skewing*, also called sector *interlacing*, picture a diskette spinning in a disk drive. The sectors in the track over which the head is positioned are passing by the head one after another—sector 1, sector 2, and so on—until the diskette has turned one complete revolution. Then the sequence repeats. A standard 8-inch diskette has 26 sectors on each track, and the disk spins at 360 rpm. One turn of the diskette takes 60/360 seconds, about 166 milliseconds per track, or 6 milliseconds per sector.

Now imagine CP/M loading a program from such a diskette. The BDOS takes a finite amount of time to read and process each sector since it reads only a single sector at a time. It has to make repeated reads to load a program. By the time the BDOS has read and loaded sector n, it will be too late to read sector n + 1. This sector will have already passed by the head and will not come around for another 166 milliseconds. Proceeding in this fashion, almost $4\frac{1}{2}$ seconds are needed to read one complete track.

This problem can be solved by simply numbering the sectors *logically* so that there are several physical sectors between each logical sector. This procedure, called *sector skewing* or *interlace*, is shown in Figure 3-4. Note that unlike physical sectors, logical sectors are numbered from 0 to 25.

Figure 3-4 shows the standard CP/M sector interlace for 8-inch, single-sided, single-density floppy diskettes. You see that logical sector 0 has six sectors between

		DPBASE:			;Base of the parameter header : (used to access the headers)
0000	1000		DW	SKEWTABLE	Pointer to logical-to-physical
0000	1000		0.	SNEWIABLE	; sector conversion table
0002	0000		DW	0	Scratch pad areas used by CP/M
	0000		DW	ō	,
0006	0000		DW	ò	
0008	2A00		DW	DIRBUF	Pointer to Directory Buffer
					; work area
000A	AA00		DW	DPBO	Pointer to disk parameter bloc
0000	B900		DW	WACD	Pointer to work area (used to
					; check for changed diskettes)
000E	C900		DW	ALVECO	<pre>;Pointer to allocation vector</pre>
		;			
		;			
		;			would normally be derived from
		;			sk parameter Block.
		;	They ar	e shown here on	ly for the sake of completeness.
003F	=	NODE	EQU	63	Number of directory entries 1
00F2	=	NOAB	EQU	242	Number of allocation blocks
		,			,
		;	Example	data definitio	ns for those objects pointed
		;	to by t	he disk paramet	er header
		;			
		SKEWTAB	LE:		;Sector skew table.
					; Indexed by logical sector
	01070D13		DB	01,07,13,19	Logical sectors 0,1,2,3;
	19050B11		DB	25,05,11,17	;4,5,6,7
	1703090F		DB	23,03,09,15	;8,9,10,11
	1502080E		DB	21,02,08,14	; 12, 13, 14, 15
	141A060C		DB	20,26,06,12	; 16, 17, 18, 19
	1218040A		DB	18,24,04,10	;20,21,22,23
0028	1016		DB	16,22	; 24, 25
		;	-		
002A		DIRBUF:		128	Directory buffer
0044		DPB0:	DS	15	;Disk parameter block
					;This is normally a table of
					; constants.
					;A dummy definition is shown
		WACD:	DC	(NODE+1)/4	; here
0080		WALLI!	DS	CNODE+17/4	;Work area to check directory :Only used for removable media
00B9					
			ne		
00B9 00C9		ALVECO:	DS	(NOAB/8)+1	;Allocation vector #0 :Needs 1 bit per allocation

Figure 3-3. Data declarations for a disk parameter header

it and logical sector 1. There is a similar gap between each of the logical sectors, so that there are six "sector times" (about 38 milliseconds) between two adjacent logical sectors. This gives ample time for the software to access each sector. However, several revolutions of the disk are still necessary to read every sector in turn. In Figure 3-4, the vertical columns of logical sectors show which sectors are read on each successive revolution of the diskette.

The wrong interlace can strongly affect performance. It is not a gradual effect, either; if you "miss" the interlace, the perceived performance will be very slow. In the example given here, six turns of the diskette are needed to read the whole track — this lasts one second as opposed to $4\frac{1}{2}$ without any interlacing. But don't imagine that you can change the interlace with impunity; files written with one interlace stay that way. You must be sure to read them back with the same interlace with which they were written.

Some disk controllers can simplify this procedure. When you format the diskette, they can write the sector addresses onto the diskette with the interlace already built in. When CP/M requests sector n, the controller's electronics wait until they see the requested sector's header fly by. They then initiate the read or write operation. In this case you can embed the interlace right into the formatting of the diskette.

Because the wrong interlace gives terrible performance, it is easy to know when you have the right one. Some programmers use the time required to format a diskette as the performance criterion to optimize the interlace. This is not good practice because under normal circumstances you will spend very little time formatting diskettes. The time spent loading a program would be a better arbiter, since far more time is spent doing this. You might argue that doing a file update would be even more representative, but most updates produce slow and sporadic disk activity. This kind of disk usage is not suitable for setting the correct interlace.

Hard disks do not present any problem for sector skewing. They spin at 3600 rpm or faster, and at that speed there simply is no interlace that will help. Some

	Logical Sector									
Physical Sector	Pass	Pass	Pass	Pass	Pass	Pas				
	1	2	3	4	5	6				
1	0									
2				13						
3			9							
4		6				22				
5		5			18					
6 7	,		ļ		18					
8	1			14						
9			10	14						
10			10			23				
11		6				2.				
12					19					
13	2									
14	-			15						
15			11							
16						24				
17		7								
18					20					
19	3]							
20				16						
21			12							
22						25				
23		8								
24					21					
25	4									
26				17						
OTE: Additional sector be		1	L		· · · · · · · · · · · · · · · · · · ·	· · · · · ·				

Figure 3-4. Physical to logical sector skewing

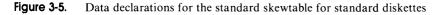
tricks can be played to improve the performance of a hard disk—these will be discussed in the section called "Special Considerations for Hard Disks," later in this chapter.

To better understand these theories, study an example of the standard interlace table, or *skewtable*. Bear in mind that the code that will access this table will first be given a *logical* sector. It will then have to return the appropriate *physical* sector.

Figure 3-5 shows the code for the skew table and the code that can be used to access the table. The table is indexed by a logical sector and the corresponding table entry is the physical sector. You can see that the code assumes that the first *logical* sector assigned by CP/M will be sector number 0. Hence there is no need to subtract 1 from the sector number before using it as a table subscript.

- **Unused Areas in the Disk Parameter Header Table** The three words shown as 0's in Figure 3-3 are used by CP/M as temporary variables during disk operations.
- **Directory Buffer (DIRBUF)** The *directory buffer* is a 128-byte area used by CP/M to store a sector from the directory while processing directory entries. You only need one directory buffer; it can be shared by all of the logical disks in the system.
- **Disk Parameter Block (DPB0)** The *disk parameter block* describes the particular characteristics of each logical disk. In general, you will need a separate parameter block for each *type* of logical disk. Logical disks can share a parameter block only if their

			_			
		SKEWTABL				;Logical sector
	01070D13		DB	01,07,		;0,1,2,3
	19050B11		DB	25,05,	11,17	;4,5,6,7
0008	1703090F		DB	23,03,	09,15	;8,9,10,11
0000	1502080E		DB	21,02,	08,14	;12,13,14,15
0010	141A060C		DB	20,26,	06,12	;16,17,18,19
0014	1218040A		DB	18,24,	04,10	;20,21,22,23
0018	1016		DB	16,22		;24,25
		;				
		;				
		;			anslate lo follows:	ogical sectors to physical
			50000		10110#51	
		,	On enti	ry, the	logical se	ctor will be transferred from
						in registers BC.
						address of the skew table
						is the skew table by looking in
		<i>.</i>			eter heade	
		2	the ur	sk param	eter neaue	r entry).
		;	0		abusies1	sector will be placed
		,		isters H		sector will be placed
			In reg	Isters n	L.	
		SECTRAN:				
001A	FR	OLC I MAN:	XCHG		م (م البار	skew table base address
001B			DAD	в		hysical sector
0018			DAD	D		
0010	/ -		-			entry in skew table
			MOV	L,M		sical sector
			MOV	н, о	:HL = Ph	nysical Sector
001D 001E			RET		:Return	



characteristics are identical. You can, for example, use a single parameter block to describe all of the single-sided, single-density diskette drives that you have in the system. However, you would need another parameter block to describe doublesided, double-density diskette drives. It is also rare to be able to share parameter blocks when a physical hard disk is split up into several logical disks. You will understand why after looking at the contents of a parameter block, described later in this chapter.

Work Area to Check for Changed Diskettes (WACD) One of the major problems that CP/M faces when working with removable media such as floppy diskettes is that the computer operator, without any warning, can open the diskette drive and substitute a different diskette. On early versions of CP/M, this resulted in the newly inserted diskette being overwritten with data from the original diskette.

With the current version of CP/M, you can request that CP/M check if the diskette has been changed. Given this request, CP/M examines the directory entries whenever it has worked on the directory and, if it detects that the diskette has been changed, declares the whole diskette to be Read-Only status and inhibits any further writing to the diskette. This status will be in effect until the next warm boot operation occurs. A warm boot occurs whenever a program terminates or a CONTROL-C is entered to the CCP, resetting the operating system.

The value of WACD is the address of a buffer, or temporary storage area, that CP/M can use to check the directory. The length of this buffer is defined (somewhat out of place) in the disk parameter block.

Allocation Vector (ALVEC0) CP/M views each disk as a set of allocation blocks, assigning blocks to individual files as those files are created or expanded, and relinquishing blocks as files are deleted.

CP/M needs some mechanism for keeping track of which blocks are used and which are free. It uses the *allocation vector* to form a *bit map*, with each bit in the map corresponding to a specific allocation block. The most significant bit (bit 7) in the first byte corresponds to the first allocation block, number 0. Bit 6 corresponds to block 1, and so on for the entire disk.

Whenever you request CP/M to use a logical disk, CP/M will *log in* the disk. This consists of reading down the file directory and, for each active entry or extent, interacting with the allocation blocks "owned" by that particular file extent. For each block number in the extent, the corresponding bit in the allocation vector is set to 1. At the end of this process, the allocation vector will accurately represent a map of which blocks are in use and which are free.

When CP/M goes looking for an unused allocation block, it tries to find one near the last one used, to keep the file from becoming too fragmented.

In order to reserve enough space for the allocation vector, you need to reserve one bit for each allocation block. Computing the number of allocation blocks is discussed in the section "Maximum Allocation Block Number," later in this chapter.

Disk Parameter Block

The disk parameter block in early versions of CP/M was built into the BDOS and was a closely guarded secret of the CP/M file system. To make CP/M adaptable to hard disk systems, Digital Research decided to move the parameter blocks out into the BIOS where everyone could adapt them. Because of the proprietary nature of CP/M's file system, you will still see several odd-looking fields, and you may find the explanation given here somewhat superficial. However, the lack of explanation in no way detracts from your ability to use CP/M as a tool.

Figure 3-6 shows the code necessary to define a parameter block for 8-inch, single-sided diskettes. This table is pointed to by—that is, its address is given in—an entry in the disk parameter header. Each of the entries shown in the disk parameter block is explained in the following sections.

Sectors Per Track This is the number of 128-byte sectors per track. The standard diskette shown in the example has 26 sectors. As you can see, simply telling CP/M that there are 26 sectors per track does not indicate whether the first sector is numbered 0 or 1. CP/M assumes that the first sector is 0; it is left to a sector translate subroutine to decipher which physical sector this corresponds to.

Hard disks normally have sector sizes larger than 128 bytes. This is discussed in the section on considerations for hard disks.

Block Shift, Block Mask, and Extent Mask These mysteriously named fields are used internally by CP/M during disk file operations. The values that you specify for them depend primarily on the size of the allocation block that you want.

Allocation block size can vary from 1024 bytes (1K) to 16,384 bytes (16K). There is a distinct trade-off between these two extremes, as discussed in the section on allocation blocks at the beginning of this chapter.

An allocation block size of 1024 (1K) bytes is suggested for floppy diskettes with capacities up to 1 megabyte, and a block size of 4096 (4K) bytes for larger floppy or hard disks.

		DPB0:			
0000	1A00		DW	26	;Sectors per track
0002	03		DB	3	;Block shift
0003	07		DB	7	;Block mask
0004	03		DB	3	;Extent mask
0005	F200		DW	242	<pre>;Max. allocation block number</pre>
0007	3F00		DW	63	Number of directory entries 1
0009	CO		DB	1100\$0000B	;Bit map for allocation blocks
000A	00		DB	0000\$0000B	: used for directory
000B	1000		DW	16	:No. of bytes in dir. check buffer
0000	0200		DW	2	:No. of tracks before directory

Figure 3-6. Data declarations for the disk parameter block for standard diskettes

34 The CP/M Programmer's Handbook

If you can define which block size you wish to use, you can now select the values for the block shift and the block mask from Table 3-1.

 Table 3-1.
 Block Shift and Mask Value

Allocation Block Size	Block Shift	Block Mask
1,024	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

Select your required allocation block size from the left-hand column. This tells you which values of block shift and mask to enter into the disk parameter block.

The last of these three variables, the *extent mask*, depends not only on the block size but also on the total storage capacity of the logical disk. This latter consideration is only important for computing whether or not there will be fewer than 256 allocation blocks on the logical disk. Just divide the chosen allocation block size into the capacity of the logical disk and check whether you will have fewer than 256 blocks.

Keeping this answer and the allocation block size in mind, refer to Table 3-2 for the appropriate value for the extent mask field of the parameter block. Select the appropriate line according to the allocation block size you have chosen. Then, depending on the total number of allocation blocks in the logical disk, select the extent mask from the appropriate column.

Table 3-2. Extent Mask Value

Number of A	llocation Blocks
1 to 255	256 and Above
0	(Impossible)
1	0
3	1
7	3
15	7
	1 to 255

Maximum Allocation Block Number This value is the *number* of the last allocation block in the logical disk. As the first block number is 0, this value is *one less* than the total number of allocation blocks on the disk. Where only a partial allocation block exists, the number of blocks is rounded down.

Figure 3-7 has an example for standard 8-inch, single-sided, single-density diskettes. Note that CP/M uses two reserved tracks on this diskette format.

Number of Directory Entries Minus 1 Do not confuse this entry with the number of files that can be stored on the logical disk; it is only the number of *entries* (minus one). Each extent of each file takes one directory entry, so very large files will consume several entries. Also note that the value in the table is *one less* than the number of entries.

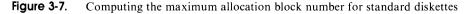
On a standard 8-inch diskette, the value is 63 entries. On a hard disk, you may want to use 1023 or even 2047. Remember that CP/M performs a sequential scan down the directory and this takes a noticeable amount of time. Therefore, you should balance the number of logical disks with your estimate of the largest file size that you wish to support.

As a final note, make sure to choose a number of entries that fits evenly into one or more allocation blocks. Each directory entry needs 32 bytes, so you can compute the number of bytes required. Make sure this number can be divided by your chosen allocation block size without a remainder.

Allocation Blocks for the Directory This is a strange value; it is not a number, but a bit map. Looking at Figure 3-6, you see the example value written out in full as a binary value to illustrate how this value is defined. This 16-bit value has a bit set to 1 for each allocation block that is to be used for the file directory.

This value is derived from the number of directory entries you want to have on the disk and the size of the allocation block you want to use. One given, or

Physical cha	racteristics:	Calculate:	
77 26 128	Tracks/ Diskette Sectors/ Track Bytes/ Sector	$-\frac{77}{-2}$	Tracks/Diskette Tracks Reserved for CP/M Tracks for File Storage
2 1024		<u>×26</u> 1950	Number of Sectors Sectors for File Storage
		×128 249,600	Bytes per Sector Bytes for File Storage
		$\frac{\div 1024}{243.75}$	Bytes/Allocation Block Total Number of
		242	Allocation Blocks Number of the last allocation block
			(rounded and based on first block being Block 0)



constant, in this derivation is that the size of each directory entry is 32 bytes.

In the example, 64 entries are required (remember the number shown is one less than the required value). Each entry has 32 bytes. The total number of bytes required for the directory thus is 64 times 32, or 2048 bytes. Dividing this by the allocation block size of 1024 indicates that two allocation blocks must be reserved for the directory. You can see that the example value shows this by setting the two most significant bits of the 16-bit value.

As a word of warning, do not be tempted to declare this value using a DW (define word) pseudo-operation. Doing so will store the value *byte-reversed*.

Size of Buffer for Directory Checking As mentioned before in the discussion of the disk parameter header, CP/M can be requested to check directory entries whenever it is working on the directory. In order to do this, CP/M needs a buffer area, called the *work area to check for changed diskettes,* or WACD, in which it can hold working variables that keep a compressed record of what is on the directory. The length of this buffer area is kept in the disk parameter block; its address is specified in the parameter header. Because CP/M keeps a compressed record of the directory, you need only provide one byte for every four directory entries. You can see in Figure 3-6 that 16 bytes are specified to keep track of the 64 directory entries.

Number of Iracks Before the Directory Figure 3-8 shows the layout of CP/M on a standard floppy diskette. You will see that the first two tracks are reserved, containing the initial bootstrap code and CP/M itself. Hence the example in Figure 3-6, giving the code for a standard floppy disk, shows two reserved tracks (the number of tracks before the directory).

This *track offset value*, as it is sometimes called, provides a convenient method of dividing a physical disk into several logical disks.

Special Considerations for Hard Disks

If you want to run CP/M on a hard disk, you must provide code and build tables that make CP/M work as if it were running on a very large floppy disk. You must even include 128-byte sectors. However, this is not difficult to do.

To adapt hard disks to the 128-byte sector size, you must provide code in the disk driver in your BIOS that will present the illusion of reading and writing 128-byte sectors even though it is really working on sectors of 512 bytes. This code is called the *blocking/deblocking* routine.

If hard disks have sector sizes other than 128 bytes, what of the number of sectors per track, and the number of tracks?

Hard disks come in all sizes. The situation is further confused by the disk controllers, the hardware that controls the disk. In many cases, you can think of the hard disk as just a series of sectors without any tracks at all. The controller, given a *relative* sector number by the BIOS, can translate this sector number into which track, read/write head (if there is more than one platter), and sector are actually being referenced.

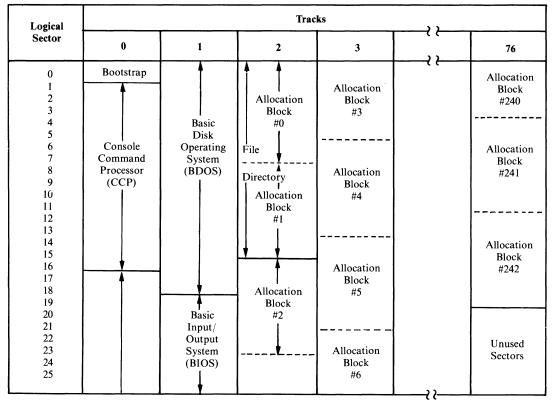


Figure 3-8. Layout of standard diskette

Furthermore, most hard disks rotate so rapidly that there is nothing to be gained by using a sector-skewing algorithm. There is just no way to read more than one physical sector per revolution; there is not enough time.

In many cases it is desirable to divide up a single, physical hard disk into several smaller, logical disks. This is done mainly for performance reasons: Several smaller disks, along with smaller directories, result in faster file operations.

The disk parameter header will have 0's for the skewtable entry and the pointer to the WACD buffer. In general, hard disks *cannot* be changed, at least not without turning off the power and swapping the entire disk drive. If you are using one of the new generation of removable hard disks, you will need to use the directory checking feature of CP/M.

The disk parameter block for a hard disk will be quite different from that used for a floppy diskette. The number of sectors per track needs careful consideration. Remember, this is the number of 128-byte sectors. The conversion from the physical sector size to 128-byte sectors will be done in the disk driver in the BIOS. If you have a disk controller that works in terms of sectors and tracks, all you need do is compute the number of 128-byte sectors on each track. Multiply the number of physical sectors per track by their size in bytes and then divide the product by 128 to give the result as the number of 128-byte sectors per physical track.

But what of those controllers that view their hard disks as a series of sectors without reference to tracks? They obscure the fact that the sectors are arranged on concentric tracks on the disk's surface. In this case, you can play a trick on CP/M. You can set the "sectors per track" value to the number of 128-byte sectors that will fit into one of the disk's physical sectors. To do this, divide the physical sector size by 128. For example, a 512-byte physical sector size will give an answer of four 128-byte sectors per "track." You can now view the hard disk as having as many "tracks" as there are physical sectors. By using this method, you avoid having to do any kind of arithmetic on CP/M's sector numbers; the "track" number to which CP/M will ask your BIOS to move the disk heads will be the *relative physical sector*. Once the controller has read this physical sector for you, you can look at the 128-byte sector number, which will be 0, 1, 2, or 3 (for a 512-byte physical sector) in order to select which 128 bytes need to be moved in or out of the disk buffer.

The block shift, block mask, and extent mask will be computed as before. Use a 4096-byte allocation block size. This will yield a value of 5 for the block shift, 31 for the block mask, and given that you will have more than 256 allocation blocks for each logical disk, an extent mask value of 1.

The maximum allocation block number will be computed as before. Keep clear in your mind whether you are working with the number of physical sectors (which will be larger than 128 bytes) or with 128-byte sectors when you are computing the storage capacity of each logical disk.

The number of directory entries (less 1) is best set to 511 for logical disks of 1 megabyte and either 1023 or 2047 for larger disks. Remember that under CP/M version 2 you cannot have a logical disk larger than 8 megabytes.

The allocation blocks for the directory are also computed as described for floppy disks.

As a rule, the size of the directory check buffer (WADC) will be set to 0, since there is no need to use this feature on hard disk systems with fixed media.

The number of tracks before the directory (track offset) can be used to divide up the physical disk into smaller logical disks, as shown in Figure 3-9.

There is no rule that says the tracks before a logical disk's directory cannot be used to contain other complete logical disks. You can see this in Figure 3-9. CP/M behaves as if each logical disk starts at track 0 (and indeed they do), but by specifying increasingly larger numbers of tracks before each directory, the logical disks can be staggered across the available space on the physical disk.

Figure 3-10 shows the calculations involved in the first phase of building disk parameter blocks for the hard disk shown in Figure 3-9. The physical characteristics are those imposed by the design of the hard disk. As a programmer, you do not have any control over these; however, you can choose how much of the physical

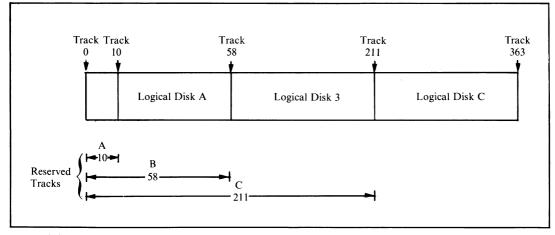


Figure 3-9. Dividing hard disks into logical disks

disk is assigned to each logical disk, the allocation block size, and the number of directory entries. You can see that logical disk A is much smaller than disks B and C, and that B and C are the same size. Disk A will be the systems disk from which most programs will be loaded, so its smaller directory size will make program loading much faster. The allocation block size for disk A is also smaller in order to reduce the amount of space wasted in partially filled allocation blocks.

Figure 3-10 also shows the calculations involved in computing the maximum allocation block number. Again, note that once the total number of allocation blocks has been computed, it is necessary to round it down in the case of any fractional components and then subtract 1 to get the maximum number (the first block being 0).

Figure 3-11 shows the actual values that will be put into the parameter blocks. It is assumed that the disk controller is one of those types that view the physical disk as a series of contiguous sectors and make no reference to tracks; the internal electronics and firmware in the controller take care of these details. For this reason, CP/M is told that each *physical* sector is a "track" in CP/M's terms. Each "track" has 512 bytes and can therefore store four 128-byte sectors. You can see this is the value that is in the sectors/"track" field.

The block shift and mask values are obtained from Table 3-1, using the allocation block size previously chosen. Then, with both the allocation block size and the maximum number of allocation blocks (see Figure 3-10), the extent mask can be obtained from Table 3-2. You can see in Figure 3-11 that extent mask values of 1 were obtained for all three logical disks even though two different allocation block sizes have been chosen, and even though disk A has less than 256 blocks and disks B and C have more.

Physical Character	ristics:		Calculate:		
364	Tracks/Di	sk			
20	Sectors/Tr	rack	A:	B: and C:	
512	512 Bytes/Sector		48	153	Tracks assigned to Disk
10,240 Bytes/Track		<u>×10,240</u>	×10,240	Bytes/Track	
			491,520	1,566,720	Bytes/Disk
			÷ 2048	÷ 4096	Bytes/Allocation Block
Chosen Logical Characteristics:		240	382.5	Number of Allocation Blocks	
		Allocation	239	381	Maximum Block Number
	Tracks	Block Size			
Reserved Area	10	\mathbf{n}/\mathbf{a}			
Disk A:	48	2048			
Disk B:	153	4096			
Disk C:	153	4096			

Figure 3-10. Computing the maximum allocation block number for a hard disk

DPBA:	DPBE	B: DPI	BC:	
4		4	4	;128-byte sectors/"track"
4		5	5	;Block shift
1	5	31	31	;Block mask
1		1	1	;Extent mask
2	39	381	381	;Max. all. block #
2	55	1023	1023	<pre>No. of directory entries</pre>
1	1110000B	11111111B	11111111B	Bit Map for allocation blocks
0	000000B	0000000B	0000000B	; used for directory
0		0	0	;No. of bytes in dir.check buffer
(10)	(58)	(211)	Actual tracks before directory
2	00	1160	4220	;"Tracks" before directory

Figure 3-11. Disk parameter tables for a hard disk

The bit map showing how many allocation blocks are required to hold the file directory is computed by multiplying the number of directory entries by 32 and dividing the product by the allocation block size. This yields results of 4 for disk A and 8 for disks B and C. As you can see, the bit maps have the appropriate number of bits set.

Since most of the hard disks on the market today do not have removable media, the lengths of the directory checking buffer are set to 0.

The number of "tracks" before the directory requires a final touch of skullduggery. Having already indicated to CP/M that each "track" has four sectors, you need to continue in the same vein and express the number of real tracks before the directories in units of 512-byte physical sectors.

As a final note, if you are specifying these parameter blocks for a disk controller that requires you to communicate with it in terms of physical tracks and 128-byte sectors, then the number of sectors per track must be set to 80 (twenty 512-byte sectors per physical track). You would also have to change the number of tracks before the directory by stating the number of physical tracks (shown in parentheses on Figure 3-11).

Adding Additional Information to the Parameter Block

Normally, some additional information must be associated with each logical disk. For example, in a system that has several physical disks, you need to identify where each *logical* disk resides. You may also want to identify some other *physical* parameters, disk drive types, I/O port numbers, and addresses of driver subroutines.

You may be tempted to extend the disk parameter header entry because there is a separate header entry for each logical disk. But the disk parameter header is exactly 16 bytes long; adding more bytes makes the arithmetic that we need to use in the BIOS awkward. The best place to put these kinds of information is to *prefix* them to the front of each disk parameter block. The label at the front of the block must be left in the same place lest CP/M become confused. Only special additional code that you write will be "smart" enough to look *in front* of the block in order to find the additional parameter information.

File Organizations

CP/M supports two types of files: sequential and random. CP/M views both types as made up of a series of 128-byte *records*. Note that in CP/M's terms, a record is the same as a 128-byte sector. This terminology sometimes gets in the way. It may help to think of 128-byte sectors as *physical* records. Applications programs manipulate *logical* records that bear little or no relation to these physical records. There is code in the applications programs to manipulate logical records.

CP/M does not impose any restrictions on the contents of a file. In many cases, though, certain conventions are used when textual data is stored. Each line of text is terminated by ASCII CARRIAGE RETURN and LINE FEED. The last sector of a text file is filled with ASCII SUB characters; in hexadecimal this is 1AH.

File Control Blocks

In order to get CP/M to work on a file, you need to provide a structure in which both you and the BDOS can keep relevant details about the file, its name and type, and so on. The file control block (FCB) is a derivative of the file directory entry, as you can see in Figure 3-12. This figure shows both a series of equates that can be used to access an entry and a series of DB (define byte) instructions to declare an example.

The first difference you will see between the file directory entry and the FCB is that the very first byte is serving a different purpose. In the FCB, it is used to specify on which disk the file is to be found. You may recall that in the directory, this byte indicates the user number for a given entry. When you are actually processing files, the current user number is set either by the operator in a command from the console or by a BDOS function call; this predefines which subset of files in the directory will be processed. Therefore, the FCB does not need to keep track of the user number.

The disk number in the FCB's first byte is stored in an odd way. A value of 0 indicates to CP/M that it should look for the file on the current default disk. This default disk is selected either by an entry from the console or by making a specific BDOS call from within a program. In general, the default disk should be preset to the disk that contains the set of programs with which you are working. This avoids unnecessary typing on the keyboard when you want to load a program.

A disk number value other than 0 represents a letter of the alphabet based on a simple codification scheme of A = 1, B = 2, and so on.

As you can see from Figure 3-12, the file name and type must be set to the required values, and for sequential file processing, the remainder of the FCB can be set to zeros. Strictly speaking, the last three bytes of the FCB (the random record number and the random record overflow byte) need not even be declared if you are never going to process the file randomly.

This raises a subtle conceptual point. Random files are only random files because *you* process them randomly. Though this sounds like a truism, what it means is that CP/M's files are not intrinsically random or sequential. What they are depends on how you choose to process them at any given point. Therefore,

0000		FCBE\$DISK	EQU	0	;Disk drive (O = default, 1=A)
0001		FCBE\$NAME	EQU	1	;File name (8 bytes)
0005	. =	FCBE\$TYP	EQU	9	;File type
					;Offsets for bits used in type
0009		FCBE\$R0	EQU	9	;Bit $7 = 1 - read$ only
000A		FCBE\$SYS	EQU	10	;Bit 7 = 1 - system status
000E	: =	FCBE\$CHANGE	EQU	11	;Bit 7 = 0 - file written to
					;
0000	; =	FCBE\$EXTENT	EQU	12	;Extent number
					;13, 14 reserved for CP/M
000F	=	FCBE\$RECUSED	EQU	15	Records used in this extent;
0010) =	FCBE\$ABUSED	EQU	16	Allocation blocks used
0020) =	FCBE\$SEQREC	EQU	32	;Sequential rec. to read/write
0021	=	FCBE\$RANREC	EQU	33	;Random rec. to read/write
0023	1 =	FCBE\$RANRECO	EQU	35	;Random rec. overflow byte (MS)
		;			
		;			
		;			
0000	00	FCB\$DISK:	DB	0	;Search on default disk drive
0001	464940454	EFCB\$NAME:	DB	'FILE	
0009	545950	FCB\$TYP:	DB	TYP'	
0000	: 00	FCB\$EXTENT:	DB	0	;Extent
1000	0000	FCB\$RESV:	DB	0,0	Reserved for CP/M
000F	- 00	FCB\$RECUSED:	DB	0	Records used in this extent
0010	000000000	OFCB\$ABUSED:	DB	0,0,0	,0,0,0,0,0 ;Allocation blocks used
0018	3 000000000	00	DB	0,0,0	,0,0,0,0,0
0020	00	FCB\$SEQREC:	DB	0	;Sequential rec. to read/write
002	0000	FCB\$RANREC:	DW	0	;Random rec. to read/write
0023	8 00	FCB\$RANRECO:	DB	0	Random rec. overflow byte (MS)

Figure 3-12. Data declarations for the FCB

while the manner in which you process them will be different, there is nothing special built into the file that predicates how it will be used.

Sequential Files

A sequential file begins at the beginning and ends at the end. You can view it as a contiguous series of 128-byte "records."

In order to create a sequential file, you must declare a file control block with the required file name and type and request the BDOS to *create* the file. You can then request the BDOS to write, "record" by "record" (really 128-byte sector by 128-byte sector) into the file. The BDOS will take care of opening up new extents as it needs to. When you have written out all the data, you must make a BDOS request to close the file.

To read an existing file, you also need an FCB with the required file name and type declared. You then make a BDOS request to open the file for processing and a series of Read Sequential requests, each one bringing in the next "record" until either your program detects an end of file condition (by examining the data coming in from the file) or the BDOS discovers that there are no more sectors in the file to read. There is no need to close a file from which you have been reading data — but *do close it*. This is not necessary if you are going to run the program only under CP/M, but it is necessary if you want to run under MP/M (the multiuser version of CP/M).

What if you need to append further information to an existing file? One option is to create a new file, copy the existing file to the new one, and then start adding data to the end of the new file. Fortunately, with CP/M this is not necessary. In the FCB used to read a file, the name and the type were specified, but you can also specify the extent number. If you do, the BDOS will proceed to open (if it can find it) the extent number that you are asking for. If the BDOS opens the extent successfully, all you need do is check if the number of records used in the extent (held in the field FCB\$RECUSED) is less than 128 (80H). This indicates the extent is not full. By taking this record number and placing it into the FCB\$SEQREC (sequential record number) byte in the FCB, you can make CP/M jump ahead and start writing from the effective end of the file.

Random Files

Random files use a simple variation of the technique described above. The main difference is that the random record number must be set in the FCB. The BDOS automatically keeps track of file extents during Read/Write Random requests. (These requests are explained more fully in Chapter 5.)

Conceptually, random files need a small mind-twist. After creating a file as described earlier, you must set the random record number in the FCB before each Write Random request. This is the two-byte value called FCB\$RANREC in Figure 3-12. Then, when you give the Write Random request to the BDOS, it will

44 The CP/M Programmer's Handbook

look at the record number; compute in which extent the record must exist; if necessary, create the directory entry for the extent; and finally, write out the data record. Using this scheme, you can dart backward and forward in the file putting records at random throughout the file space, with CP/M creating the necessary directory entries each time you venture into a part of the file that has not yet been written to.

The same technique is used to read a file randomly. You set the random record number in the FCB and then give a system call to the BDOS to open the correct extent and read the data. The BDOS will return an error if it cannot find the required extent or if the particular record is nonexistent.

Problems lie in wait for the unwary. Before starting to do any random reading or writing, you must open up the file at extent 0 even though this extent may not contain any data records. For a new file, this can be done with the Create File request, and for an existing file with the normal Open File request. If you create a *sparse* file, one that has gaps in between the data, you may have some problems manipulating the file. It will appear to have several extents, each one being partially full. This will fool some programs that normally process sequential files; they don't expect to see a partial extent except at the end of a file, and may treat the wrong spot as the end. Functions of the CCP Editing the CCP Command Line Built-In Commands Program Loading Base Page Memory Dumps of the Base Page Processing the Command Tail Available Memory Communicating with the BIOS Returning to CP/M



The Console Command Processor (CCP)

The Console Command Processor processes commands that you enter from the console. As you may recall from the brief overview in Chapter 2, the CCP is loaded into memory immediately below the BDOS. In practice, many programs deliberately overwrite the CCP in order to use the memory it normally occupies. This gives these programs an additional 800H bytes (2K bytes).

When one of these "transient programs" terminates, it relinquishes control to the BIOS, which in turn reloads a fresh copy of the CCP from the system tracks of the disk back into memory and then transfers control to it. Consequently, the CCP leads a sporadic existence—an endless series of being loaded into memory, accepting a command from you at the console, being overwritten by the program you requested to be loaded, and then being brought back into memory when the program terminates.

This chapter discusses what the CCP does for you in those brief periods when it is in memory.

Functions of the CCP

Simply put, once the CCP has control of the machine, so do you. The CCP announces its presence by displaying a prompt of two characters: a letter of the alphabet for the current default disk drive and a "greater than" sign. In the example A>, the A tells you that the default disk drive is currently set to be logical drive A, and the ">," that the message was output by the CCP.

Once you see the prompt, the CCP is ready for you to enter a command line. A command line consists of two major parts: the name of the command and, optionally, some values for the command. This last part is known as the *command tail*.

The command itself can be one of two things: either the name of a file or the name of one of the frequently used commands built into the CCP.

If you enter the name of one of the built-in commands, the CCP does not need to go out to the disk system in order to load the command for execution. The executable code is already inside the CCP.

If the name of the command you entered does not match any of the built-in commands (the CCP has a table of their names), the CCP will search the appropriate logical disk drive for a file with a matching name and a file type of "COM" (which is short for command). You do not enter ".COM" when invoking a command — the CCP assumes a file type of "COM."

If you do not precede the name of the COM file with a logical disk drive specification, the CCP will search the current default drive. If you have prefixed the COM file's name with a specific logical drive, the CCP will look only on that drive for the program. For example, the command MYPROG will cause the CCP to look for a file called "MYPROG.COM" on the current default drive, whereas C:MYPROG would make the CCP search only on drive C.

If you enter a command name that matches neither the CCP's built-in command table nor the name of any COM file on the specified disk, the CCP will output the command name followed by a question mark, indicating it is unable to find the file.

Editing the CCP Command Line

The CCP uses a line buffer to store what you type until you strike either a CARRIAGE RETURN or a LINE FEED. If you make an error or change your mind, you can modify the incomplete command, even to the point of discarding it.

You edit the command line by entering *control characters* from the console. Control characters are designated either by the combination of keys required to generate them from the keyboard or by their official name in the ASCII character set. For example, CONTROL-J is also known as CARRIAGE RETURN or CR.

Whenever CP/M has to represent control characters, the convention is to indicate the "control" aspect of a character with a caret ("^"). For example, CONTROL-A will appear as "^A", CONTROL-Z as "^Z", and so on. But if you press the CONTROL key with the normal shift key and the "6" key, this will produce a CONTROL-^ or "^^". The representation of control keys with the caret is only necessary when outputting to the console or the printer—internally, these characters are held as their appropriate binary values.

CONTROL-C: Warm Boot If you enter a CONTROL-C as the first character of a command line, the CCP will initiate a warm boot operation. This operation resets CP/M completely, including the disk system. A fresh copy of the CCP is loaded into memory and the file directory of the current default disk drive is scanned, rebuilding the allocation bit map held in the BIOS (as discussed in Chapter 3).

The only time you would initiate a warm boot operation is after you have changed a diskette (or a disk, if you have removable media hard disks). Thus, CP/M will reset the disk system.

Note that a CONTROL-C only initiates a warm boot if it is the first character on a command line. If you enter it in any other position, the CCP will just echo it to the screen as " C ". If you have already entered several characters on a command line, use CONTROL-U or CONTROL-X to cancel the line, and then use CONTROL-C to initiate a warm boot. You can tell a warm boot has occurred because there will be a noticeable pause after the CONTROL-C before the next prompt is displayed. The system needs a finite length of time to scan the file directory and rebuild the allocation bit map.

CONTROL-E: Physical End-of-Line The CONTROL-E command is a relic of the days of the teletype and terminals that did not perform an automatic carriage return and line feed when the cursor went off the screen to the right. When you type a CONTROL-E, CP/M sends a CARRIAGE RETURN/LINE FEED command to the console, but does not start to execute the command line you have typed thus far. CONTROL-E is, in effect, a *physical* end-of-line, not a *logical* one.

As you can see, you will need to use this command only if your terminal either overprints (if it is a hard copy device) or does not wrap around when the cursor gets to the right-hand end of the line.

CONTROL-H: Backspace The CONTROL-H command is the ASCII backspace character. When you type it, the CCP will "destructively" backspace the cursor. Use it to correct typing errors you discover before you finish entering the command line. The last character you typed will disappear from the screen. The CCP does this by sending a three-character sequence of backspace, space, backspace to the console. The CCP ignores attempts to backspace over its own prompt. It also takes care of backspacing over control characters that take two character positions on the line. The CCP sends the character sequence backspace, backspace, space, space, backspace, backspace, erasing both characters.

- **CONTROL-J: Line Feed/CONTROL-M: Carriage Return** The CONTROL-J command is the ASCII LINE FEED character; CONTROL-M is the CARRIAGE RETURN. Both of these characters terminate the command line. The CCP will then execute the command.
- **CONTROL-P: Printer Echo** The CONTROL-P command is used to turn on and off a feature called *printer echo*. When it is turned on, every character sent to the console is also sent to CP/M's list device. You can use this command to get a hard copy of information that normally goes only to the console.

CONTROL-P is a "toggle." The first time you type CONTROL-P it turns on printer echo; the next time you type CONTROL-P it turns off printer echo. Whenever CP/M does a warm boot, printer echo is turned off.

There is no easy way to know whether printer echo is on or off. Try typing a few CARRIAGE RETURNS, and see whether the printer responds; if it does not, type CONTROL-P and try again.

One of the shortcomings in most CP/M implementations is that the printer drivers (the software in the BIOS that controls or "drives" the printer) do not behave very intelligently if the printer is switched off or not ready when you or your program asks it to print. Under these circumstances, the software will wait forever and the system will appear to be dead. So if you "hang" the system in this way when you type a CONTROL-P, check that the printer is turned on and ready. Otherwise, you may have to reset the entire system.

CONTROL-R: Repeat Command Line The CONTROL-R command makes the CCP repeat or retype the current input line. The CCP outputs a "#" character, a CARRIAGE RETURN/LINE FEED, and then the entire contents of the command line buffer. This is a useful feature if you are working on a teletype or other hard copy terminal and have used the RUB or DEL characters. Since these characters do not destructively delete a character, you can get a visually confusing line of text on the terminal. The CONTROL-R character gives you a fresh copy of the line without any of the logically deleted characters cluttering it up. In this way you can see exactly what you have typed into the command line buffer.

See the discussion of the RUB and DEL characters for an example of CONTROL-R in use.

CONTROL-S: Stop Screen Output The CONTROL-S command is the ASCII XOFF (also called DC3) character; XOFF is an abbreviation for "Transmit Off." Typing CONTROL-S will temporarily stop output to the console. In a standard version of

CP/M, the CCP will resume output when *any* character is entered (including another CONTROL-S) from the console. Thus, you can use CONTROL-S as a toggle switch to turn console output on and off.

In some implementations of CP/M, the console driver itself (the low-level code in the BIOS that controls the console) will be maintaining a communication protocol with the console; therefore, a better way of resuming console output after pausing with a CONTROL-S is to use CONTROL-Q, the ASCII XON or "Transmit On" character. Entering a CONTROL-Q instead of relying on the fact that *any* character may be used to continue the output is a fail-safe measure.

The commands CONTROL-S and CONTROL-Q are most useful when you have large amounts of data on the screen. By "riding" the CONTROL-S and CONTROL-Q keys, you can let the data come to the screen in small bursts that you can easily scan.

CONTROL-U or CONTROL-X: Undo Command Line The commands CONTROL-U and CONTROL-X perform the same function: They erase the current partially entered command line so that you can undo any mistakes and start over. The CONTROL-U command was originally intended for hard copy terminals. The CCP outputs a "#" character, then a CARRIAGE RETURN/LINE FEED, and then some blanks to leave the cursor lined up and ready for you to enter the next command line. It leaves what you originally entered in the previous line on the screen. The CONTROL-X command is more suited to screens; the CCP destructively backspaces to the beginning of the command line so that you can reenter it.

RUB or DEL: Delete Last Character The rubout or delete function (keys marked RUB, RUBOUT, DEL, or DELETE) nondestructively deletes the last character that you typed. That is, it deletes the last character from the command line buffer and echoes it back to the console.

Here is an example of a command line with the last few characters deleted using the RUB key:

```
A>RUN PAYROLLLLORYAPSALES
```

You can see that the command line very quickly becomes unreadable. If you lose track of what are data characters and what has been deleted, you can use CONTROL-R to get a fresh copy of what is in the command line buffer.

The example above would then appear as follows:

A>RUN PAYROLLLLORYAPSALES# RUN SALES_

The "#" character is output by the CCP to indicate that the line has been

repeated. The "_" represents the position of the cursor, which is now ready to continue with the command line.

Built-In Commands

When you enter a command line and press either CARRIAGE RETURN or LINE FEED, the CCP will check if the command name is one of the set of built-in commands. (It has a small table of command names embedded in it, against which the entered command name is checked.) If the command name matches a built-in one, the CCP executes the command immediately.

The next few sections describe the built-in commands that are available; however, refer to *Osborne CP/M User Guide*, second edition by Thom Hogan (Berkeley: Osborne/McGraw-Hill, 1982) for a more comprehensive discussion with examples of the various forms of each command.

X: - Changing Default Disk Drives The default drive is the currently active drive that CP/M uses for all file access whenever you do not nominate a specific drive. If you wish to change the default drive, simply enter the new default drive's identifying letter followed by a colon. The CCP responds by changing the name of the disk that appears in the prompt line.

On hard disks, this simple operation may take a second or two to complete because the BDOS, requested by the CCP to log in the drive, must read through the disk directory and rebuild the allocation vector for the disk. If you have a diskette or a disk that is removable, changing it and performing a warm boot has the same effect of refreshing CP/M's image of which allocation blocks are used and which are available. It takes longer on a hard disk because, as a rule, the directories are much larger.

DIR – Directory of Files In its simplest form, the DIR command displays a listing of the files set to Directory status in the current user number (or file group) on the current default drive. Therefore, when you do not ask for any files after the DIR command, a file name of "*.*" is assumed. This is a total wildcard, so all files that have not been given System status will be displayed. This is the only built-in command where an omitted file name reference expands to "all file names, all file types."

You can display the directory of a different drive by specifying the drive in the same command line as the DIR command.

You can qualify the files you want displayed by entering a unique or ambiguous file name or extension. Only those files that match the given file name specification will be displayed, and even then, only those files that are not set to System status will appear on the screen. (The standard CP/M utility program STAT can be used to change files from SYS to DIR status.)

Another side effect of the DIR command and files that are SYS status is best illustrated by an example. Imagine that the current logical drive B has two files on it called SYSFILE (which has SYS status) and NONSYS (which does not). Look at the following console dialog, in which user input is underlined:

 B: DIR
 SYSFILE does not show

 B: DIR
 JUNK<</td>

 B>DIR
 JUNK

 NO FILE
 JUNK does not exist

 B>DIR
 SYSFILE

 B>____
 SYSFILE

Do you see the problem? If a file is not on the disk, the CCP will display NO FILE (or NOT FOUND in earlier versions of CP/M). However, if the file *does* exist but is a SYS file, the CCP does not display it because of its status; nor does the CCP say NO FILE. Instead it quietly returns to the prompt. This can be confusing if you are searching for a file that happens to be set to SYS status. The only safe way to find out if the file does exist is to use the STAT utility.

ERA – Erase a File The ERA command logically removes files from the disk (*logically* because only the file directory is affected; the actual data blocks are not changed).

The logical delete changes the first byte of each directory entry belonging to a file to a value of 0E5H. As you may recall from the discussion on the file directory entry in Chapter 3, this first byte usually contains the file user number. If it is set to 0E5H, it marks the entry as being deleted.

ERA makes a complete pass down the file directory to logically delete all of the extents of the file.

Unlike DIR, the ERA command does not assume "all files, all types" if you omit a file name. If it did, it would be all too easy to erase all of your files by accident. You must enter "*.*" to erase all files, and even then, you must reassure the CCP that you really want to erase all of them from the disk. The actual dialog looks like the following:

```
A><u>era b:*.*<cr></u>
ALL (Y/N)?<u>y<cr></u>
A>_
```

If you change your mind at the last minute, you can press "n" and the CCP will not erase any files.

One flaw in CP/M is that the ERA command only asks for confirmation when you attempt to erase all of your files using a name such as "*.*" or "*.???". Consider the impact of the following command:

A><u>ERA</u> <u>*.C??<cr></u> A>_

The CCP with no hesitation has wiped out all files that have a file type starting with the letter "C" in the current user number on logical disk A.

If you need to use an ambiguous file name in an ERA command, check which files you will delete by first using a STAT command with exactly the same ambiguous file name. STAT will show you all the files that match the ambiguous name, even those with SYS status that would not be displayed by a DIR command.

There are several utility programs on the market with names like UNERA or WHOOPS, which take an ambiguous file name and reinstate the files that you may have accidentally erased. A design for a version of UNERASE is discussed in Chapter 11.

If you attempt to erase a file that is not on the specified drive, the CCP will respond with a NO FILE message.

REN — Rename a File The REN command renames a file, changing the file name, the file type, or both. In order to rename, you need to enter two file names, the new name and the current file name.

To remember the correct name format, think of the phrase new = old. The actual command syntax is

A><u>ren</u> <u>newfile.typ=oldfile.typ<cr></u> A>_

You can use a logical disk drive letter to specify on which drive the file exists. If you specify the drive, you only need to enter it on one of the file names. If you enter the drive with both file names, it must be the same letter for both.

Unlike the previous built-in command, REN cannot be used with ambiguous file names. If you try, the CCP echoes back the ambiguous names and a question mark, as in the following dialog:

```
A>ren chap*.doc=chapter*.doc<cr>
CHAP*.DOC=CHAPTER*.DOC?
A>_
```

If the REN command cannot find the old file, it will respond NO FILE. If the new file already exists, the message FILE EXISTS will be displayed. If you receive a FILE EXISTS message and want to check that the new file does exist, remember that it is better to use the STAT command than DIR. The extant file may be declared to be SYS status and therefore will not appear if you use the DIR command.

TYPE – Type a Text File The TYPE command copies the specified file to the console. You cannot use ambiguous file names, and you will need to press CONTROL-S if the file has more data than can fill one screen. With the TYPE command, the data in the file will fly past on the screen unless you stop the display by pressing CONTROL-S. Be careful, because if you type any other character, the TYPE command will abort and return control to the CCP.

Once you have had time to see what is displayed on the screen, you can press CONTROL-Q to resume the output of data to the console. With standard CP/M implementations, you will discover that any character can be used to restart the flow of data; however, use CONTROL-Q as a fail-safe measure. CONTROL-S (X-OFF) and CONTROL-Q (X-ON) conform to the standard protocol which should be used.

If you need to get hard copy output of the contents of the file, you should type a CONTROL-P command before you press the CARRIAGE RETURN at the end of the TYPE command line.

As you may have inferred, the TYPE command should only be used to output ASCII text files. If for some reason you use the TYPE command with a file that contains binary information, strange characters will appear on the screen. In fact, you may program your terminal into some state that can only be remedied by turning the power off and then on again. The general rule therefore is *only* use the TYPE command with ASCII text files.

SAVE – Save Memory Image on Disk The SAVE command is the hardest of the CCP's commands to explain. It is more useful to the programmer than to a typical end user. The format of this command is

A><u>SAVE n FILENAME.TYP<cr></u> A>_

The SAVE command creates a file of the specified name and type (or overwrites an existing file of this name and type), and writes into it the specified number n of memory pages. A page in CP/M is 256 (100H) bytes. The SAVE command starts writing out memory from location 100H, the start of the Transient Program Area (TPA). Before you use this command, you will normally have loaded a program into the TPA. The SAVE command does just what its name implies: It saves an image of the program onto a disk file.

More often than not, when you use the SAVE command the file type will be ".COM." With the file saved in this way, the CCP will be able to load and execute the file.

USER – Change User Numbers As mentioned before, the directory of each logical disk consists of several directories that are physically interwoven but logically separated by the user number. When you use a specific user number, those files that were created when you were in another user number are logically not available to you.

The USER command provides a way for you to move from one user number to another. The command format is

 $A>USER n < er> A>_$

where n can be any number from 0 to 15. Any other number will provoke the CCP to echoing back your entry, followed by a question mark.

54 The CP/M Programmer's Handbook

But once you have switched back and forth between user numbers several times, it is easy to become confused about which user number you are in. The STAT command can be used to find the current user number. If you are in a user number that does not make a copy of STAT available to you however, all you can do is use the USER command to set yourself to another user number. You cannot find out which user number you were in; you can only tell the system the user number you want to go to.

In the custom BIOS systems discussed later, there is a way of displaying the current user number each time a warm boot occurs. If you are building a system in which you plan to utilize CP/M's user number features, you should give this display of the current user number serious thought. If you are in the wrong user number and erase files, you can create serious problems.

Some implementations of CP/M have modified the CCP so that the prompt shows the current user number as well as the default drive (similar to the prompt used in MP/M). However, this use of a nonstandard CCP is not a good practice. As a rule, customization should be confined to the BIOS.

Program Loading

The first area to consider when loading a program is the first 100H bytes of memory, called the *base page*. Several fields — units in this area of memory — are set to predetermined values before a program takes control.

To aid in this discussion, imagine a program called COPYFILE that copies one file to another. This program expects you to specify the source and destination file names on the command line. A typical command would read

A>copyfile tofile.typ fromfile.typ display

Notice the word "display." COPYFILE will, if you specify the "display" option, output the contents of the source file ("fromfile.typ") on the console as the transfer takes place.

When you press the CARRIAGE RETURN key at the end of the command line, the CCP will search the current default drive ("A" in the example) and load a file called COPYFILE.COM into memory starting at location 100H. The CCP then transfers control to location 100H—just past the base page—and COPYFILE starts executing.

Base Page

The base page normally starts from location 0000H in memory, but where there is other material in low memory addresses, it may start at a higher address. Figure 4-1 shows the assembly language code you will need to access the base page. RAM is assumed to start at location 0000H in this example.

0000	_	D 4 M		•	
0000	=	RAM	EQU	0	;Start of RAM (and the base page) ;You may need to change this to ; some other value (e.g. 4300H)
0000		;	ORG	RAM	;Set location counter to RAM base
0000		WARMBOOT:	DS	3	;Contains a JMP to warm boot entry ; in BIOS Jump vector table
0002	=	; BIOSPAGE	EQU	RAM+2	;BIOS Jump vector page
		,			
0003		IOBYTE: ;	DS	1	;Input/output redirection byte
0004		CURUSER:	DS	1	;Current user (bits 7-4)
0004	=	CURDISK ;	EQU		;Default logical disk (bits 3-0)
0005		BDOSE:	DS	3	;Contains a JMP to BDOS entry
0007	=	TOPRAM ;	EQU	BDOSE+2	;Top page of usable RAM
00050	:	;	ORG	RAM+5CH	;Bypass unused locations
0050		FCB1:	DS	16	;File control block #1 ;Note: if you use this FCB here ; you will overwrite FCB2 below.
006C		; FCB2:	DS	16	;File control block #2 ;You must move this to another
					; place before using it
0080		7	ORG	RAM+80H	;Bypass unused locations
		COMTAIL:			Complete command tail
0080		COMTAIL COUNT:	DS	1	;Count of the number of chars ; in command tail (CR not incl.)
0081		COMTAIL\$CHARS:	DS	127	;Characters in command tail ; converted to uppercase and
					; without trailing carriage ret.
		;			
0080			ORG	RAM+SOH	Redefine command tail area;
0080		DMABUFFER:	DS	128	;Default "DMA" address used ; as a 128-byte record buffer
		;			
0100			ORG	RAM+100H	;Bypass unused locations
		TPA:			;Start of transient program area ; into which programs are loaded.

Figure 4-1. Base page data declarations

Some versions of CP/M, such as the early Heathkit/Zenith system, have ROM from location 0000H to 42FFH. Digital Research, responding to market pressure, produced a version of CP/M that assumed RAM starting at 4300H. If you have one of these systems, you must add 4300H to all addresses in the following paragraphs *except* for those that refer to addresses at the top of memory. These will not be affected by the presence of ROM in low memory.

The individual values used in fields in the base page are described in the following sections.

Warmboot The three-byte *warmboot* field contains an instruction to jump up to the high end of RAM. This JMP instruction transfers control into the BIOS and triggers a warm boot operation. As mentioned before, a warm boot causes CP/M to reload the CCP and rebuild the allocation vector for the current default disk. If you need to cause a warm boot from within one of your assembly language programs, code

JMP 0 ;Warm Boot

BIOSPAGE The BIOS has several different entry points; however, they are all clustered together at the beginning of the BIOS. The first few instructions of the BIOS look like the following:

JMP	ENTRY1			
JMP	ENTRY2			
JMP	ENTRY3	;and	50	on

Because of the way CP/M is put together, the first jump instruction *always* starts on a page boundary. Remember that a page is 256 (100H) bytes of memory, so a page boundary is an address where the least significant eight bits are zero. For example, the BIOS jump vector (as this set of JMPs is called) may start at an address such as F200H or E600H. The exact address is determined by the size of the BIOS.

By looking at the BIOSPAGE, the most significant byte of the address in the warmboot JMP instruction, the page address of the BIOS jump vector can be determined.

IOBYTE CP/M is based on a philosophy of separating the *physical* world from CP/M's own *logical* view of the world. This philosophy also applies to the character-oriented devices that CP/M supports.

The IOBYTE consists of four two-bit fields that can be used to assign a physical device to each of the logical ones. It is important to understand that the IOBYTE itself is just a passive data structure. Actual assignment occurs only when the physical device drivers examine the IOBYTE, interpreting its contents and selecting the correct physical drive for the cooperation of the BIOS. These device drivers are the low-level (that is, close to machine language) code in the BIOS that actually interfaces and controls the physical device.

The four logical devices that CP/M knows about are

1. *The console.* This is the device through which you communicate with CP/M. It is normally a terminal with a screen and a keyboard. The console is a bidirectional device: It can be used as a source for information (input) and a destination to which you can send information (output).

In CP/M terminology, the console is known by the symbolic name of "CON:". Note the ":"—this differentiates the device name from a disk file that might be called "CON."

2. The list device. This is normally a printer of some sort and is used to make hard copy listings. CP/M views the printer as an output device only. This creates problems for printers that need to tell CP/M they are busy, but this

problem can be remedied by adding code to the low-level printer driver. CP/M's name for this logical device is "LST:".

3. The paper tape reader. It is unusual to find a paper tape reader in use today. Originally, CP/M ran on an Intel Microcomputer Development System called the MDS-800, and this system had a paper tape reader. This device can be used only as a source for information.

CP/M calls this logical device "RDR:".

4. The paper tape punch. This, too, is a relic from CP/M's early days and the MDS-800. In this case, the punch can be used only for output. The logical device name used by CP/M is "PUN:".

The physical arrangement of the IOBYTE fields is shown in Figure 4-2.

Each two-bit field can take on one of four values: 00, 01, 10, and 11. The particular value can be interpreted by the BIOS to mean a specific physical device, as shown in Table 4-1.

Although the actual interpretation of the IOBYTE is performed by the BIOS, the STAT utility can set the IOBYTE using the logical and physical device names, and PIP (Peripheral Interchange Program) can be used to copy data from one device to another. In addition, you can write a program that simply changes the

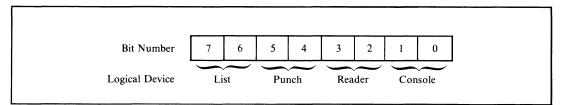


Figure 4-2. Arrangement of the IOBYTE

Logical Device	Physical Device								
	00	01	10	11					
Console (CON:) Reader (RDR:) Punch (PUN:) List (LST:)	TTY: TTY: TTY: TTY: TTY:	CRT: PTR: PTP: CRT:	BAT: UR1: UP1: LPT:	UC1: UR2: UP2: UL1:					

contents of the IOBYTE. But be careful: Changes in the IOBYTE take effect immediately.

The values in the IOBYTE have the following meanings:

Console (CON:)

- 00 Teletype driver (TTY:) This driver is assumed to be connected to a hard copy device being used as the main console.
- 01 CRT driver (CRT:) The driver is assumed to be connected to a CRT terminal.
- 10 Batch mode (BAT:) This is a rather special case. It is assumed that appropriate drivers will be called so that console input comes from the logical reader (RDR:) and console output is sent to the logical list device (LST:).
- 11 User defined console (UC1:) Meaning depends on the individual BIOS implementation. If, for example, you have a high-resolution graphics screen, you could arrange for this setting of the IOBYTE to direct console output to it. You might make console input come in from some graphic tablet, joystick, or other device.

Reader (RDR:)

- 00 Teletype driver (TTY:) This refers to the paper tape reader device that was often found on teletype consoles.
- 01 Paper tape reader (PTR:) This presumes some kind of high-speed input device connected to the system. Modern systems rarely have such a device, so this setting is often used to connect the logical reader to the input side of a communications line.
- 10 User defined reader #1 (UR1:)
- 11 User defined reader #2 (UR2:) Both of these settings can be used to direct the physical driver to some other specialized devices. These values are included only because they would otherwise have been unassigned. They are rarely used.

Punch (PUN:)

- 00 Teletype driver (TTY:) This refers to the paper tape punch that was often found on teletype consoles.
- 01 Paper tape punch (PTP:)

This presumes that there is some kind of high-speed paper tape punch connected to the system. Again, this is rarely the case, so this setting is often used to connect the logical punch to the output side of a communications line.

- 10 User defined punch #1 (UP1:)
- 11 User defined punch #2 (UP2:) These two settings correspond to the two user defined readers, but they are practically never used.

List (LST:)

- 00 Teletype driver (TTY:) Output will be printed on a teletype.
- 01 CRT driver (CRT:) Output will be directed to the screen on a CRT terminal.
- 10 Line printer driver (LPT:) Output will go to a high-speed printing device. Although the name *line printer* implies a specific type of hardware, it can be any kind of printer.
- 11 User defined list device (UL1:)Whoever writes the BIOS can arrange for this setting to cause logical list device output to go to a device other than the main printer.

To repeat: The IOBYTE is not actually used by the main body of CP/M. It is just a passive data structure that can be manipulated by the STAT utility. Whether the IOBYTE has any effect depends entirely on the particular BIOS implementation.

- **CURUSER** The CURUSER field is the most significant four bits (high order nibble) of its byte. It contains the currently selected user number set by the CCP USER command, by a specific call to the BDOS, or by a program setting this nibble to the required value. This last way of changing user numbers may cause compatibility problems with future versions of CP/M, so use it only under controlled conditions.
- **CURDISK** The CURDISK field is the least significant four bits of the byte it shares with CURUSER. It contains a value of 0 if the current disk is A:, 1 if it is B:, and so on. The CURDISK field can be set from the CCP, by a request to the BDOS, or by a program altering this field. The caveat given for CURUSER regarding compatibility also applies here.
- **BDOSE** This three-byte field contains an instruction to jump to the entry point of the BDOS. Whenever you want the BDOS to do something, you can transfer the request to the BDOS by placing the appropriate values in registers and making a CALL to this JMP instruction. By using a CALL, the return address will be

placed on the stack. The subsequent JMP to the BDOS does not put any additional information onto the stack, which operates on a last-in, first-out basis; so when the system returns from the BDOS, it will return directly to your program.

- **TOPRAM** Because the BDOS, like the BIOS, starts on a page boundary, the most significant byte of the address of the BDOS entry tells you in which page the BDOS starts. You must subtract 1 from the value in TOPRAM to get the highest page number that you can use in your program. Note that when you use this technique, you assume that the CCP will be overwritten since it resides in memory just below the BDOS.
- **FCB1 and FCB2** As a convenience, the CCP takes the first two parameters that appear in the command tail (see next section), attempts to parse them as though they were file names, and places the results in FCB1 and FCB2. The results, in this context, mean that the logical disk letter is converted to its FCB representation, and the file name and type, converted to uppercase, are placed in the FCB in the correct bytes. In addition, any use of "*" in the file name is expanded to one or more question marks. For example, a file name of "abc*.*" will be converted to a name of "ABC?????" and type of "???".

Notice that FCB2 starts only 16 bytes above FCB1, yet a normal FCB is at least 33 bytes long (36 bytes if you want to use random access). In many cases, programs only require a single file name. Therefore, you can proceed to use FCB1 straight away, not caring that FCB2 will be overwritten.

In the case of the COPYFILE program example on previous pages, two file names are required. Before FCB1 can be used, the 16 bytes of FCB2 must be moved into a skeleton FCB that is declared in the body of COPYFILE itself.

COMTAIL The command tail is everything on the command line *other* than the command name itself. For example, the command tail in the COPYFILE command line is shown here:

A>copyfile tofile.type fromfile.typ display

The CCP takes the command tail (converted to uppercase) and stores it in the COMTAIL area.

COMTAIL\$COUNT This is a single-byte binary count of the number of characters in the command tail. The count does *not* include a trailing CARRIAGE RETURN or a blank between the command name and the command tail. For example, if you enter the command line

A>PRINT ABC*.*

the COMTAIL\$COUNT will be six, which is the number of characters in the string "ABC*.*".

- **COMTAIL\$CHARS** These are the actual characters in the command tail. This field is not blank-filled, so you must use the COMTAIL\$COUNT in order to detect the end of the command tail.
- **DMA\$BUFFER** In Figure 4-1, the DMA\$BUFFER is actually the same area of memory as the COMTAIL. This is a space-saving trick that works because most programs process the contents of the command tail before they do any disk input or output. The DMA\$BUFFER is a sector buffer (hence it has a length of 128 bytes). The use of the acronym DMA (direct memory access) refers back to the Intel MDS-800. This system had hardware that could move data to and from diskettes by going directly to memory, bypassing the CPU completely. The term is still used even though you may have a computer system that does not use DMA for its disk I/O. You can substitute the idea of "the address to/from which data is read/written" in place of the DMA concept.

You can request CP/M to use a DMA address other than DMA\$BUFFER, but whenever the CCP is in control, the DMA address will be set back here.

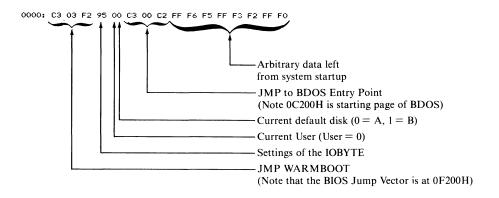
This is the *transient program area* into which the CCP loads programs. The TPA extends up to the base of the BDOS.

The TPA is also the starting address for the memory image that is saved on disk whenever you use the CCP SAVE command.

Memory Dumps of the Base Page

The following are printouts showing the contents of the base page (the first 100H bytes of memory) as the COPYFILE program will see it.

This is an example of the first 16 bytes of memory:



TPA

The command line, as you recall, was

A>copyfile tofile.typ fromfile.typ display

The FCB1 and FCB2 areas will be set by the CCP as follows:

Logi	cal	Dis	k								L	ogi	cal	Dis	k	
0050:		5 4		46 F									ł			
0060:	49 I	4C			20	54 T	59 Y	50 P	00	00	00	00	00	46 F	52 R	4F 0
0070:	4D M	46 F	49 I	40 L	45 E	54 T	59 Y	50 P	00 •	00 •	00 •	00 •	00 •	F2 •	34 4	FЗ •

Since the logical disks were not specified in the file names in the command line, the CCP has set the disk code in both FCB1 and FCB2 to 00H, meaning "use the default disk." The file name and type have been converted to uppercase, separated, and put into the FCBs in their appointed places.

The complete command tail has been stored in COMTAIL as follows:

You can see that the command tail length is 01FH (31 decimal). This is followed immediately by the command tail characters themselves. Note that the command tail stops at location 9FH. The remainder of the data that you can see is the residue of some previous directory operation by the CCP. You can see the file name CRCK.COM in a directory entry, followed by several 0E5Hs that are unused directory space.

Finally, at location 0100H are the first two bytes of the program.

Processing the Command Tail

One of the first problems facing you if you write a program that can accept parameters from the command tail is to process the command tail itself, isolating each of the parameters. You should use a standard subroutine to do this. This subroutine splits the command line into individual parameters and returns a count of the number of parameters, as well as a pointer to a table of addresses. Each address in this table points in turn to a null-byte-terminated string. Each parameter is placed in a separate string.

Figure 4-3 contains the listing of this subroutine, CTP (Command Tail Processor).

ļ							
	0100			ORG	100H		
	0100 0103	CD3601 00	START:	CALL	СТР	;Test b	ed for CTP
				; Rema	inder of	your pr	ogram
			;	This su	broutine	breaks	the command tail apart, placing
			;	each va	alue in a	separa	te string area.
			;	Return	paramete		
			;				r (Z flag set)
			?				mber of parameters addresses
			;				dress points to a null-byte-
			;				ted parameter string.
			;				meters are specified, then A = TMP
			;				meter is too long, then A = PTL
			;				ints to the first character of the ng parameter in the COMTAIL area.
			;				is parameter in the contrine areas
	0080		COMTAIL			80H	;Command tail in base page
	0080		COMTAIL	COUNT			Count of chars. in command tail
	0001		CTP\$TMP CTP\$PTL			1 2	;Too many parameters error code ;Parameter too long error code
	0002	-	:		200	2	Farameter too long error code
			PTABLE:			;Table	of pointers to parameters
		0001		DW		; Param	
	0106 0108			DW DW		; Param	
	0108	2001		DW	· -	; Param	parameter addresses here
	010A	0000		DW		; Termi	
			;	_			
			;		ter strin		that unused parameters appear
			;		null stri		that unused parameters appear
			;				is a O and is used to detect
			;		neter tha		
		0001010101010101010101010101010101010101					1,1,1,1,1,0 ;Param. 1 & terminator
		000101010101					1,1,1,1,1,0 ;Param. 2 & terminator 1,1,1,1,1,0 ;Param. 3 & terminator
							parameter strings here
			;				-
			CTP:				Main entry point <<<<<
		210401 0E00		LXI MVI	H, PTABLE C.O		;HL -> table of addresses ;Set parameter count
		348000		LDA	COMTAILS		Character count
	013E			ORA	A		Check if any params.
	013F			RZ			Exit (return params. already set)
	0140 0141			PUSH	H B.A		Save on top of stack for later
		218100		LXI	H, COMTAI		;B = COMTAIL char. count ;HL -> Command tail chars.
	••••						

Figure 4-3. Command Tail Processor (CTP)

CTPSNEXTP: 0145 E3 0145 E3 0146 5E 0146 5E 0146 5E 0146 5E 0147 23 0147 23 0147 23 0147 23 0149 55 0149 55 0149 7A 0144 B3 0 07A 015 0144 B3 0 07A 07A 015 0144 B3 0 07A 07A 07A 07A 07A 07A 07A 07A 07A 0			OTONIEVT	-		Next secondary land
0146 5E HOV E.H ifter of stack = COMPAIL ptr. 0147 23 INX H ilpdate address pointer 0148 56 HOV A.D iDE to paramater string (or is 0) 0149 7A HOV A.D iDE to paramater string (or is 0) 0148 7B HOV A.D iDE to paramater string (or is 0) 0148 7B HOV A.D iDE to paramater string (or is 0) 0148 7B HOV A.D iDE to paramater string (or is 0) 0148 7B ONA E iDE to point for the stress in ing 0148 7B INX H ilpdate command tail 0149 7E HL >> net byte in command tail iftro for stack-=undate addr. ptr. 0147 FE HNX H ilpdate command tail 0150 7E HOV A.M ilpdate command tail 0150 7E HOV INX H ilpdate command tail 0150 7E HOV A.M ilpdate command tail intremain 0150 7E HOV INX H ilpdate command tail ptr. 0151 23 INX D ilfdate command tail ptr. ilfdate command tail		AE 50				Next parameter loop
Old4 5EMOVE.M;Det LS byte of param. addr.Ol49 7AMOVD.M;Get MS byte of param. addr.Ol49 7AMOVA.D;Get MS byte of addr.Ol49 7AMOVA.D;Get MS byte of addr.Ol44 B3DRAE.Combine MS and LS byteOl44 E3JNXH;HE address pointerOl45 CAR001JZCTPSTMPX;Too many parameters-rexitOl44 E3JNXH;HE address pointers-rexitOl45 CAR001JZCTPSTMPX;Too many parameters-rexitOl47 E3JNXH;HE address pointers-rexitOl47 E3JNXH;HE address pointers-rexitOl47 E3JNXH;Get next parameter stringCTPSKIPB:CTPSKIPB:;CTPSKIPB:CTPSKIPB:CTPSKIPB:;CtPSKIPB:Ol50 7EMOVA.M;Get next parameter stringOl50 7EMOXA.M;Get next parameter stringOl50 7EMOXA.M;Get next parameter stringOl50 12StranetXXD;Increment parameter stringOl50 13INXD;Update commant string ptr.Ol55 7EORAA;Store in parameter stringOl55 87ORAAStore in parameter stringOl55 7EORAAiStore in parameter stringOl55 7EORAAiStore in parameter stringOl56 623INXH;Update commant string ptr.Ol56 7700ORAAiStore in parameter string	0	140 E3				
Ol47 23INKH; Update address pointerOl48 55MOVD,H:Get MS byte of param. addr. :DE -> Parameter string (or is 0)Ol44 62ORAE; Combine MS and LS byteOl44 62ORAE; Combine MS and LS byteOl46 62ORAE; Combine MS and LS byteOl47 63NTHLHUChate pointer to rest address : HL -> heat byte in command tail : DE -> first byte of next parameter hyteOl50 7EMYXA,M; Got next parameter stringOl52 05DCRB: Check if characters still remainOl53 7A7301MBCTP*SKIPB : YEs, so skip blanksOl53 64CTINXA,MOl53 65CTINXOl55 13DCRIncrement parameter stringOl55 14LDAXDOl55 15LDAXDOl56 7FORACOl56 16CPIOl57 17INXOl58 16LDAXOl58 16LDAXOl58 17DOl56 18LDAXOl57 19Yes, so skip blanksOl56 19ORAOl56 14LDAXOl56 15CPIOl56 16CPIOl56 7FORAOl56 7FORAOl56 7FORAOl56 7FORAOl57 19YEAOl56 7FORAOl56 7FORAOl57 7FYEAOl56 7FORAOl57 7FORAOl56 7FORAOl57 7F <td></td> <td>144 55</td> <td></td> <td>MOU</td> <td>EM</td> <td></td>		144 55		MOU	EM	
Olds 56MOVD, M; DE -> Parameter string Cor is on ; DE -> Parameter stringOlds 7AMOVA, D; DE -> Optime MS and LS byteOlds CasonJZCTPSTMPX; Too many parameters == exitOlds CasonJZCTPSTMPX; Too many parameters == exitOlds CasonJZCTPSTMPX; Too many parameters == exitOlds CasonJE-> comtail; DE -> comtailToo cason, DE -> first byte of next parameter stringOlso 7EMOVA, M; Get next parameter stringOlso 7EIDA, M; DE -> first byte of next parameter stringOlso 7EIDA, M; Get next parameter stringOlso CasonMDCTPSKIPB:; DE -> first byte of next parameter stringOlso CasonMDCTPSKIPB; Check if characters still remainOlso CasonMDCTPSKIPB; YA, so exitOlso CasonMDCTPSKIPB; YA, so exitOlso CasonMDCTPSKIPB; YA, so exitOlso CasonJNXD; Under parameter string PT.Olso CasonJNXD; Under param						
0149 7A 0144 B3 0145 CA8001 0145 CA8001 0147 C3 0147 C4 0147 C4 0148						
Ol49 7AMOV OTAA.DJOEt copy of MS byte of addr. JCombine MS and LS byteOl44 B3 Ol48 CA8001JZCTP\$TMPX Too many parameters—exit IUdate pointer to next address HL-> comtail TOO for stack—update addr. ptr. JH -> next byte in command tail State to state the state of the state					2711	
Ol4A E3ORAE;Combine MS and LS byte0148 CA8001JZCTPSTMPX;Too many parameters-e-wit0144 E3INXH;Uddate pointer to next address0147 E3XTHL:Top of stack-update addr. ptr.iAt this point, we have; HL -> contail:DE -> first byte of next parameter string0150 7EINXCTPSKIPB::DE -> first byte of next parameter string0151 72INX0152 05DCR0153 7EINX0153 7EINX0153 7EINX0153 7EINX0153 7EINX0154 7E20CFI0155 7EINX0155 7EOCR0155 7EINX0155 7EINX0155 7EINX0155 7EINX0156 7FINX0157 7EStar0158 0CINX0159 7EYes, so skip blanks0150 7EINX0150 7EINX0151 12INX0152 13INX0155 7EMGV0156 7EMGV0160 CA7A01JZ0165 7EMGV0163 AFXRA0165 7EMGV0165 7EMGV0166 7FMGV0167 7EMGV0168 7F201JZ0168 7F201JZ0168 7F201JT0168 7F201JT0168 7F201JT0169 7F2CTPSTL0169 7F2CTPSTL0169 7F2	0	149 7A	1	MOV	A. D	
0148 CA8001JZCTP\$TMPX:Too many parametersexit014F E3INXH;Udde pointer to next address014F E3XTHL;HL -> contail014F E3XTHL;HL -> contail0147 E3INXpeak byte in command tail; HL -> next byte in command tail;DE -> first byte of next parameter string0150 7EMOVA,M0151 7EMOVA,M0152 7EMOVA,M0153 FA7301MBCTP*KIPB:0153 FA7301CTP*KIPB0158 CAS001JZCTP*KIPB0158 CAS001JZCTP*KIPB0158 CAS001JZCTP*KIPB0158 CAS001JZCTP*KIPB0158 CAS001JZCTP*KIPB0158 CAS001JZStore in parameter string0159 13INXD0150 14LDAX D;Check next byte0155 15CTP*NEXTC:;Darameter too long exit0156 16LDAX D;Check if tarameter dail0157 87ORA A;Float a 00-byte at end of param.0163 AFXRA A;Float a 00-byte at end of param.0164 72301JHCTPX0165 787ORA A;Sloet in parameter still remain0166 747301JZCTP*NEXTC0167 75JOHGTPX0168 77301JHCTPX0169 77301JHCTP*NEXTC0160 74501JZCTP*NEXTC0170 3F7CAM0173 3F7XRA A0174 817GTP						
OIAF E3YTHLJTHLJHL ->:contail (Top of stack-update addr. ptr. is the tin command tail i bE -> first byte of next parameter stringOIS0 7EMOV (Stack-update addr. ptr. is DE -> first byte of next parameter stringOIS0 7EMOV (Stack-update addr. ptr. is DE -> first byte of next parameter stringOIS0 7EMOV (Stack-update addr. ptr. is DE -> first byte of next parameter stringOIS0 7EMOV (Stack-update addr. ptr. is DE -> first byte of next parameter stringOIS0 7EMOV (Stack-update addr. ptr. is DC COIS1 23INX INX (Stack-update addr. ptr. is CTP48KIPB (Stack-update addr. ptr. is	0	14B CA8001		JZ	CTP\$TMPX	;Too many parametersexit
$\begin{array}{c c c c c c c c c c c c c c c c c c c $	0	14E 23		INX	н	;Update pointer to next address
i HL -> hext byte in command taili HL -> hext byte in command taili HL -> hext byte of next parameter string0150 7EMOV0151 23INX0152 05DCR0153 FA7301MB0153 FA7301MB0153 FE20CPI0156 FE20CPI0157 FE20CPI0158 0CINR0159 0CCTP\$KIPB0150 12STAX0150 12CTP\$KIPB0150 13INX0151 14LDAX0152 14LDAX0152 15CTP\$FTLX0155 16CTP\$FTLX0156 17ORA0157 18CTP\$FTLX0158 02CTP\$FTLX0159 13INX0159 14LDAX0159 15INX0150 16CA70010151 14LDAX0152 15ORA0155 16CA70010155 17ORA0165 17ORA0165 16CTP\$FTLX164 12STAX0165 16DCR17CTP1066 171066 171016 121017 121016 121018 14119119110112 <trr>113<td>0</td><td>14F E3</td><td></td><td>XTHL</td><td></td><td>;HL -> comtail</td></trr>	0	14F E3		XTHL		;HL -> comtail
$ \begin{array}{c} \begin{array}{c} \begin{array}{c} \ \ \ \ \ \ \ \ \ \ \ \ \ $;Top of stackupdate addr. ptr.
$ \begin{array}{c} ; DE \rightarrow first byte of next parameter string \\ CTP$SKIPS; \\ O150 7E & MOV A,M & Get next parameter hyte \\ O151 23 & INX H & Ulpdate command tail ptr. \\ O152 05 & DCR B & Check if characters still remain \\ O153 FA7301 & MB & CTPX & tho, so exit \\ O156 FE20 & CPI & ' & Check if blank \\ O158 0C & INR & C & florement parameter counter \\ CTPSNETC: & florement parameter string \\ O150 13 & INX D & tlpdate parameter string ptr. \\ O152 14 & LDAX D & tlpdate parameter string ptr. \\ O155 FB & ORA A & flore next byte \\ O156 FB & ORA A & flore next byte \\ O156 FB & ORA A & flore next byte \\ O156 FB & ORA A & flore next byte \\ O156 FB & ORA A & flore next byte \\ O160 CA7A01 & JZ & CTP$PTLX & flore next character from tail \\ O163 FE & MOV A,M & flore next character from tail \\ O164 12 & STAX D & flore next character from tail \\ O164 FF & XRA & D & flore next character from tail \\ O164 FF & XRA & D & flore next character from tail \\ O164 FF & MOV A,M & flore next character from tail \\ O164 FF & NC & N & flore still remain \\ O165 FE & MOV A,M & flore next character from tail \\ O164 FF & XRA & D & flore next character from tail \\ O166 FA7301 & UM & CTPX & tho, so exit \\ O167 CS & DCR & B & flore next character from tail \\ O168 FA7301 & UM & CTPX & tho so exit \\ O170 C35001 & JHP & CTPNEXTC & tho, so store it in parameter \\ O177 AF & XRA & the O & Z-flag set \\ trong trong trong trong trong trong trong trong trong \\ trong trong trong trong trong trong exit \\ O177 ASE02 & NVI & A,CTP$PTL trong trong exit trong trong \\ trong trong trong trong exit \\ O177 ASE02 & NVI & A,CTP$PTL trong trong exit \\ O177 ASE02 & NVI & A,CTP$PTL trong trong exit \\ O177 ASE02 & NVI & A,CTP$PTL trong trong exit \\ O177 ASE02 & NVI & A,CTP$PTL trong trong exit \\ O177 ASE02 & NVI & A,CTP$PTL trong exit \\ O177 ASE01 & JMP & CTPCX & tCommon exit \\ O178 B7 & ORA & trong parameters exit \\ O178 B7 & ORA & trong parameters exit \\ O178 B7 & ORA & trong parameter trong trong exit \\ O178 B7 & ORA & trong parameters exit \\ O178 B7 & ORA & trong parameter trong t$						
CTPSKIPS: 0150 7E MOV A,M ;Get next parameter byte 0151 23 INX H ;Update command tail ptr. 0152 FA7301 MB CTPX ;No, so exit 0153 FA7301 MB CTPX ;No, so exit 0156 FE20 CPI '' ;Check if blank 0158 CA5001 JZ CTP4SKIPB ;Yes, so skip blanks 0158 0C INR C CTPSNEXTC: 015C 12 STAX D ;Store in parameter string 015D 13 INX D ;Update parameter string 015D 13 CTP4FTLX ; 015E 1A LDAX D ;Check next byte 015F B7 ORA A ;CTP4FTLX ; 0166 CA7A01 JZ CTP4FTLX ; 0166 CA7A01 JZ CTP4FTLX ; 0166 CA7A01 JZ CTP4FTLX ; 0166 7E MOV A,M ;Get next character from tail 0166 FE20 CPI '' ;Gommon exit in parameter 0167 C35 DCR B ;CTPX ;No, so store it in parameter 0168 FE20 CPI '' ;Gommon exit code 177 4E1 POP H ;Balance stack 1077 3AF XRA A ;Ensure Z-flag set appropriately 0178 B7 ORA A ;Ensure Z-flag set appropriately 177 3E0 MVI A,CTP4FTL ;Get error code 177 3E0 MVI A,CTP4FTL ;Gommon exit 180 3E01 JMP CTPCX ;Common exit						
0150 7EMOVA,M;Get next parameter hyte0151 23INXH;Update command tail ptr.0152 05DCRB;Check if characters still remain0153 FA7301MBCTFX;No, so exit0156 FE20CPI'.';Check if blank0158 CASSO1JZCTP\$SKIPB;Yes, so skip blanks0158 CASSO1JZCTP\$SKIPB;Yes, so skip blanks0150 12INRC;Increment parameter string0150 13INXD;Update parameter string ptr.0157 14LDAXD;Check next byte0158 157ORAA;Float a OO-byte at end of param.0156 1575MVVA,M;Get next character from tail0156 75DCRB;Check if characters still remain0158 7605DCRB;Check if characters still remain0156 765DCRB;Check if characters still remain0158 7620CPI ';Check if parameter terminator0158 7620JHCTPX;Normal exit0158 77XRAA;A = 0 & Z-flag set170 C3501JHCTPNEXTC;Normal exit0159 77RAA;A = 0 & Z-flag set170 774 81POPH;Balance stack0173 AFXRAA;A = 0 & Z-flag set appropriately177 3E02MVIA,CTP\$FTL;Parameter too long exit0170 C37401LXIH,PTABLE;DE -> offending parameter0177 3E02MVIA,CTP\$FTH			CTRACKID	D .	; DE -/ TIPSU D	Syte of next parameter string
0151 23INXHjUpdate command tail ptr.0152 05DCRB;Check if characters still remain0153 FA7301MBCTPX;No, so exit0156 FE20CPI'.';Check if blank0158 CASS01JZCTP\$SKIPB;Yes, so skip blanks0158 CASS01JZCTP\$NEXTC;;Store in parameter string0151 13INXD;Dedke next byte0155 14LDAXD;Check if terminator0156 CA7A01JZCTP\$PTLX;Float a OD-byte at end of param.0164 12STAXD;Store in param. string0165 7EMOXA,M;Get next character from tail0166 23INXH;Update command tail pointer0168 FA7301JMCTPX;No, so setit0169 FE20CPI';Check if parameter terminator0160 CAS501JZCTPNEXTC;No, so store it in param. string0167 C35C01JMPCTPNEXTC;No, so store it in param. string0170 C35C01JMPCTPNEXTC;Normal exit0173 AFXRAA;A = 0 & Z-flag setiCTP\$PTLX;CTP\$NEXTC;Somon exit0173 BFQCAA;Ensure Z-flag set appropriately0174	<u>م</u>	150 7F			A M	Get next parameter hyte
0152<05DCRB:Check if characters still remain0153 <fa7301< td="">MBCTP;No, so exit0156<fe20< td="">CPI'/';Check if blank0158<cas001< td="">JZCTP*SKIPB;Yes, so skip blanks0158<0C</cas001<></fe20<></fa7301<>						
0153 FA7301MBCTPX;No, so exit0156 FE20CFI:Check if blank0158 CA5001JZCTP\$SKIPB;Yes, so skip blanks0158 0CINR:Store in parameter counterCTP\$NEXTC::Store in parameter string ptr.0150 13INN:Update parameter string ptr.0155 14LDAX:Store in parameter string of parameter0157 E7GRAa0160 CA7A01JZCTP\$PTLX0163 FEMOVA,M0164 12STAXpitore in param. string0165 7EMOVA,M0166 23INXH0168 FA7301JMCTPX0168 FA7301JMCTPX0168 FA7301JM0168 FA7301JM0170 C35C01JM0170 C35C01JM0173 AFXRA7CTP\$NEXTC7:Normal exit0173 AFXRA7CTP\$NEXTC0174 E1POP0175 210401LXI0176 23520CPI0177 CF9RET0178 AFXRA0179 C9RET0170 C3701LXI0177 CF9Store code0177 CF9YCH0178 B7CTP\$PTLX:0179 C9RET0170 C37401JM0170 C37401Store code0170 C37401JM0170 C37401Store code0170 C37401JM0170 C37401JM0170 C37401JM01						
0156 FE20CPI'/'; Check if blank0158 CA5001JZCTP\$KIPB; Yes, so skip blanks0158 0CINRC; Increment parameter counter0150 12STAXD; Store in parameter string ptr.0150 13INXD; Update parameter string ptr.0155 14LDAXD; Check if terminator0156 15ORAA; Check if terminator0157 87ORAA; Store in parameter too long exit0163 AFXRAA; Float a 00-byte at end of param.0164 12STAXD; Store in parameter for tail0165 7EMOVA,M; Get next character from tail0165 7EMOVA,M; Update command tail pointer0165 7EMOVA,M; Check if characters still remain0165 7EMOVA,M; Check if characters still remain0165 7EMOVJACTPx0165 7EDCRB; Check if parameter terminator0165 7EMOVA,M; Get next character from tail0165 7EMOVJACTPx0165 7EMOVA,M0165 7EMOVA,M0165 7EMOVA,M0165 7EMOVA,M0165 7EMOVA,M0165 7EJACTPx0165 7EJACTPx0165 7ECPI;0165 7EDCRB0165 7ECPI0166 7E20CPI(TPSNEXTP <td< td=""><td></td><td></td><td></td><td></td><td></td><td></td></td<>						
0158CASO01JZCTP\$SKIPB;Yes, so skip blanks01580CINR;Increment parameter counter015012STAXp015113INXp015214LDAXp015514LDAXp01557ORAr0160CA7A01JZCTP\$PTLX0163FXRAr016412STAXp016412STAXp01657EMOV016423INXH01657EMOV016663INX016675DCR016705DCR016775DCR0168FA201JM0168FA201JM0168FA201JM0168CTP177NC0160CA501JZ0170CS501JM7CTP\$NEXTC7NO1070CS5011070CS5011070CS5011173AF1173AF1173AF1174POP1174POP117521040111752104011174FR11752104011174A11752104011174A117521040111752104011176217117721						
$\begin{array}{c c c c c c c c c c c c c c c c c c c $				JZ	CTP\$SKIPB	
015C 12 STAX D ;Store in parameter string 015D 13 INX D ;Update parameter string ptr. 015E 1A LDAX D ;Check next byte 015F B7 ORA A ;CTP\$PTLX ;Parameter too long exit 0163 AF XRA A ;Float a 00-byte at end of param. 0164 12 STAX D ;Store in parameter from tail 0165 7E MOV A, M ;Get next character from tail 0166 23 INX H ;Update command tail pointer 0167 05 DCR B ;Check if characters still remain 0168 FA7301 JH CTPX ;No, so exit 0160 CA4501 JZ CTP\$NEXTP ;Yes, so move to next parameter 0170 C35C01 JHP CTP\$NEXTC ;No, so store it in param. 0170 C35C01 JHP CTP\$NEXTC ;No, so store it in param. 0177 E1 POP H ;Balance stack 0174 E1 POP H ;Balance stack 0175 B7 ORA A ;A CTP\$TLX ;Parameter too long exit 0170 C35C01 JHP CTPSNEXTC ;No, so exit code 0174 E1 POP H ;Balance stack 0175 210401 LXI H, PTABLE ;Return ptr. to param. addr. table 0178 B7 ORA A ;Ensure Z-flag set appropriately 0170 C37601 JHP CTPSTLX ; 0173 AF ;CTPSTLX; ;Parameter too long exit 0176 SD ORA A ;Ensure Z-flag set appropriately 0177 C59 RET 0170 C37401 JHP CTPCX ;Common exit 0170 C37401 JHP CTPCX ;Common exit ; 1070 C37401 JHP CTPCX ;Common exit ; 1070 C37401 JHP CTPCX ;Common exit ; 1070 MAY Parameter sexit ; 1070 MAY Parameter sexit ; 1070 MAY Parameters exit ; 1070 MAY Parameters exit ; 1070 MAY Parameters exit ; 1070 MAY Parameters exit ; 1070 MAY PARAMENT ; ; 1070 MAY Parameters exit ; 1070 MAY PARAMENT ; ; 1070	0	15B OC		INR	С	;Increment parameter counter
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					_	
OISE IALDAXD;Check next byteOISE IALDAXD;Check next byteOISE IAUISF B7ORAAOISO CA7A01JZCTP\$PTLX;Parameter too long exitOISA AFXRAA;Float a 00-byte at end of param.OIAG 12STAXD;Store in param. stringOIAG 12STAXD;Store in param. stringOIAG 12MOVA,M;Get next character from tailOIAG 23INXH;Update command tail pointerOIAG 705DCRB;Check if characters still remainOIAG 705DCRB;Check if parameter terminatorOIAG 705DCRB;Check if parameter terminatorOIAG 705DCRB;Check if parameter terminatorOIAG 705DCRB;Check if parameter terminatorOIAG 705JZCTP\$NEXTP;Yes, so move to next parameterOIAG 704JZCTP\$NEXTC;No, so store it in param. stringOIAG 705CA4501JZCTP*NEXTCOIAG 705CTPCX;Common exit;Common exitCTPX:;TPA;Normal exitOIAG 705CTPCX;Common exitOIAG 705CTPWIOIAG 705CTP;Set error codeOIAG 705CTPYCHOIAG 705CTP;Common exitOIAG 705CTPYCHOIAG 705CTPOIAG 705CTPOIAG 705CTPOIAG 705CTP </td <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
015F B7 ORA A ; Check if terminator 0160 CA7A01 JZ CTPBTLX ; Parameter too long exit 0163 AF XRA A ; Float a Oo-byte at end of param. 0164 12 STAX D ; Store in param. string 0165 7E MOV A,M ; Get next character from tail 0166 23 INX H ; Update command tail pointer 0167 05 DCR B ; Check if characters still remain 0168 FA7301 JM CTPX ; No, so exit 0168 FA7301 JM CTPX ; No, so exit 0168 CA4501 JZ CTP\$NEXTC ; No, so store it in param. string 0170 C35C01 JMP CTP\$NEXTC ; No, so store it in param. string 7 CTPX: ; Normal exit 0173 AF XRA A ; A = 0 & Z-flag set 7 CTPCX ; CTP\$NEXTC ; Normal exit code 0174 E1 POP H ; Balance stack 0175 210401 LXI H, PTABLE ; Return ptr. to param. addr. table 0179 C9 RET 7 CTP\$NEXTC ; Set error code 0174 3E02 MVI A, CTP\$PTL ; Set error code 0174 3E02 MVI A, CTP\$PTL ; Set error code 0174 3E02 MVI A, CTP\$PTL ; Common exit 0170 C37401 JMP CTPCX ; Common exit 0170 C37401 JMP CTPCX ; Common exit 0170 S210401 LXI H, PTABLE ; Return ptr. to param. addr. table 0178 B7 ORA A ; Ensure Z-flag set appropriately 7 CTP\$PTLX: ; fParameter too long exit 0174 3E02 MVI A, CTP\$PTL ; Set error code 0174 C1P STA01 JMP CTPCX ; Common exit 0180 3E01 MVI A, CTP\$TMP ; Set error code 0182 C37401 JMP CTPCX ; Common exit 7 CTP\$CY ; Common exit 7 CTP\$CY ; Common exit 7 CTP\$CY ; COMAN PARAMETER ; COMMON exit 7 CTP\$CY ; CTP\$TMPX: ; Common exit 7 CTP\$CY ; Common exit 7 CTP\$CY ; COMAN PARAMETER ; Set error code 1080 3E01 MVI A, CTP\$TMP ;					-	
0160CA7A01JZCTP\$PTLX;Parameter too long exit0163AFXRAA;Float a OO-byte at end of param.016412STAXD;Store in param. string01657EMOVA,M;Get next character from tail0165016623INXH016705DCRB;Check if characters still remain0168FA7301JMCTPX;No, so exit0160CA4501JZCTP\$NEXTP;Yes, so move to next parameter0170C35C01JMPCTP\$NEXTC;No, so store it in param. string0173AFXRA;A = 0 & Z-flag set7CTPX;Common exit code0174E1POPH0175210401LXIH,PTABLE0178B7ORAA0173SEC2XCHG0173SEC2XCHG0173SEC2KCHG0174E2SCHFT0175210401LXI0173BFORA0174SEC20175CTP\$TLX:;Parameter too long exit0170C374010170JMPCTP\$X;Parameter too long exit0174SEC2175SEC3176YE177SEC4178YE179SEC4174SEC2174SEC2176YE177YE178YE <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
$\begin{array}{cccccccccccccccccccccccccccccccccccc$						
016412STAXDiStore in param. string01657EMOVA,M:Get next character from tail016623INXH:Update command tail pointer016705DCRB;Check if characters still remain0168FA7301JMCTPX;No, so exit0168FE20CPI'';Check if parameter terminator0160CA4501JZCTP\$NEXTP;Ves, so move to next parameter0170C35C01JMPCTP\$NEXTC;No, so store it in param. stringictry:ictry::Normal exitictry::CTPCX:Normal exitictry::CTPCX:Common exit code0174E1POPH0175210401LXI0178B7ORA0178B7ORA0173SECMVI0174SECYearameter too long exit0175CTPTLX::Parameter too long exit0176CTPSTLX::Parameter too long exit0177:SEC:Offending parameter0170:ST401JMPCTPSTMPX::Too many parameters exit01803E01MVI0182C37401JMPif:Set error codeif:Set error codeif:Set error codeif:Set error codeif:Set error codeif:Set error code:Set error code:Set error code:Set error code:Set error cod						
$\begin{array}{cccccccccccccccccccccccccccccccccccc$						
0166<23INXH;Update command tail pointer0167<05						
0167 05DCR BB;Check if characters still remain No, so exit No, so exit0168 F47301JMCTPX Y;No, so exit yNo, so exit0168 FE20CPI''0160 CA4501JZCTP\$NEXTP YNO, so store it in parameter yNo, so store it in parameter0170 C35C01JMP0173 AFXRA7CTPCX CTPCX174 E1POP YOF RET0175 210401LXI RET0178 B7 0179 C9QRA RET0173 3E02MVI XCHG0174 3E02 017D C37401MVI JMP0170 CTPSX CTPSTLX:;Parameter too long exit yCammeter yCommon exit0170 C37401JMP CTPCX0170 C37401JMP ZCTPSTTL0180 3E01 0182 C37401MVI JMP0180 3E01 0182 C37401MVI JMP180 3E01 0182 C37401 <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
$\begin{array}{cccccccccccccccccccccccccccccccccccc$						
0168 FE20 CPI / / (for the construction of th						
0170 C3SC01 JMP CTP\$NEXTC ;No, so store it in param. string i CTPX: ;Normal exit 0173 AF XRA A ;A = 0 & Z-flag set i CTPCX ;Common exit code 0174 E1 POP H ;Balance stack 0175 210401 LXI H,PTABLE ;Return ptr. to param. addr. table 0178 B7 ORA A ;Ensure Z-flag set appropriately 0179 C9 RET ;Common exit code i CTP\$PTLX: ;Parameter too long exit ;DE -> offending parameter 0170 C37401 JMP CTPCX ;Common exit i CTP\$TTMPX: ;Too many parameters exit ;DE -> offending parameters exit 0180 3E01 MVI A,CTP\$TMP ;Set error code i JMP CTPCX ;Common exit ;Common exit				CPI		Check if parameter terminator
image: construction of the second state of the second s	l õ	16D CA4501		JZ	CTP\$NEXTP	;Yes, so move to next parameter
0173 AF XRA A ;A = 0 & Z-flag set ; CTPCX ;Common exit code 0174 E1 POP H ;Balance stack 0175 210401 LXI H,PTABLE ;Return ptr. to param. addr. table 0178 B7 ORA A ;Ensure Z-flag set appropriately 0179 C9 RET ; CTP\$PTLX: ;Parameter too long exit 017A 3E02 MVI A,CTP\$PTL ;Set error code 017D C37401 JMP CTPCX ;Common exit ; CTP\$TMPX: ;Too many parameters exit 0180 3E01 JMP CTPCX ;Common exit ;	0	170 C35C01		JMP	CTP\$NEXTC	;No, so store it in param. string
0173 AF XRA A ;A = 0 & Z-flag set ; CTPCX ;Common exit code 0174 E1 POP H ;Balance stack 0175 210401 LXI H,PTABLE ;Return ptr. to param. addr. table 0178 B7 ORA A ;Ensure Z-flag set appropriately 0179 C9 RET ; CTP\$PTLX: ;Parameter too long exit 017A 3E02 MVI A,CTP\$PTL ;Set error code 017D C37401 JMP CTPCX ;Common exit ; CTP\$TMPX: ;Too many parameters exit 0180 3E01 JMP CTPCX ;Common exit ;			;			
image: construct of the state of the st						
CTPCX ;Common exit code 0174 E1 POP H ;Balance stack 0175 210401 LXI H,PTABLE ;Return pt. to param. addr. table 0178 B7 ORA A ;Ensure Z-flag set appropriately 0179 C9 RET ;CTP\$PTLX: ;Parameter too long exit 017A 3E02 MVI A,CTP\$PTL ;Set error code 017C EB XCHG ;DE -> offending parameter 017D C37401 JMP CTPCX ;Common exit ; CTP\$TMPX: ;Too many parameters exit 0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;	0	173 AF		XRA	A	;A = 0 & Z-flag set
0174 E1 POP H ;Balance stack 0175 210401 LXI H,PTABLE ;Return ptr. to param. addr. table 0178 B7 ORA A ;Return ptr. to param. addr. table 0179 C9 RET ; CTP\$PTLX: ;Parameter too long exit 017A 3E02 MVI A,CTP\$PTL ;Set error code 017D C37401 JMP CTPCX ;Common exit ; CTP\$TMPX: ;Too many parameters exit 0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;						-Common quit and
0175 210401 LXI H,PTABLE ;Return ptr. to param. addr. table 0178 B7 ORA A ;Ensure Z-flag set appropriately 0179 C9 RET CTP\$PTLX: ;Parameter too long exit 017A 3E02 MVI A,CTP\$PTL ;Set error code 017C EB XCHG ;DE -> offending parameter 017D C37401 JMP CTPCX ;Common exit ; CTP\$TMPX: ;Too many parameters exit 0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit				000	u	
0178 B7 ORA A ;Ensure Z-flag set appropriately 0179 C9 RET CTP\$PTLX: ;Parameter too long exit 017A 3E02 MVI A,CTP\$PTL ;Set error code 017C EB XCHG ;DE -> offending parameter 017D C37401 JMP CTPCX ;Common exit ;CTP\$TMPX: ;Too many parameters exit 0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;						
0179 C9 RET CTP\$PTLX: ;Parameter too long exit 017A 3E02 MVI A,CTP\$PTL ;Set error code 017C EB XCHG ;DE -> offending parameter 017D C37401 JMP CTPCX ;Common exit ; 0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;					· · · · · · · · · · · · · · · · · · ·	
; CTP\$PTLX: ;Parameter too long exit O17A 3E02 MVI A,CTP\$PTL ;Set error code O17C EB XCHG ;DE -> offending parameter O17D C37401 JMP CTPCX ;Common exit ; CTP\$TMPX: ;Too many parameters exit O180 3E01 MVI A,CTP\$TMP ;Set error code O182 C37401 JMP CTPCX ;Common exit ;					n	Armader F itad ber abbiobitatett
017A 3E02 MVI A,CTP\$PTL ;Set error code 017C EB XCHG ;DE -> offending parameter 017D C37401 JMP CTPCX ;Common exit ; CTP\$TMPX; ;Too many parameters exit 0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;	ľ	.,, .,		· · • •		
017A 3E02 MVI A,CTP\$PTL ;Set error code 017C EB XCHG ;DE -> offending parameter 017D C37401 JMP CTPCX ;Common exit ; CTP\$TMPX: ;Too many parameters exit 0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;	1		CTP\$PTLX	:		;Parameter too long exit
017D C37401 JMP CTPCX ;Common exit ; CTP\$TMPX: ;Too many parameters exit 0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;	0	17A 3E02			A, CTP\$PTL	
; CTP\$TMPX: ;Too many parameters exit 0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;	0	17C EB		XCHG		
0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;	0	17D C37401		JMP	CTPCX	;Common exit
0180 3E01 MVI A,CTP\$TMP ;Set error code 0182 C37401 JMP CTPCX ;Common exit ;			; CTR&TMPY			Too many parameters exit
0182 C37401 JMP CTPCX ;Common exit ;		190 2501				
	ľ	102 00/401		0.1	004	,
	0	185		END	START	

Figure 4-3. Command Tail Processor (CTP) (continued)

Available Memory

Many programs need to use all of available memory, and so very early in the program they need to set the stack pointer to the top end of the available RAM. As mentioned before, the CCP can be overwritten as it will be reloaded on the next warm boot.

Figure 4-4 shows the code used to set the stack pointer. This code determines the amount of memory in the TPA and sets the stack pointer to the top of available RAM.

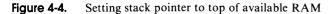
Communicating with the BIOS

If you are writing a utility program to interact with a customized BIOS, there will be occasions where you need to make a *direct* BIOS call. However, if your program ends up on a system running Digital Research's MP/M Operating System, you will have serious problems if you try to call the BIOS directly. Among other things, you will crash the operating system.

If you need to make such a call and you are aware of the dangers of using direct BIOS calls, Figure 4-5 shows you one way to do it.

Remember that the first instructions in the BIOS are the jump vector — a sequence of JMP instructions one after the other. Before you can make a direct call, you need to know the *relative page offset* of the particular JMP instruction you want to go to. The BIOS jump vector always starts on a page boundary, so all you need to know is the least significant byte of its address.

0007	-	TOPRAM	EQU	/	Most significant byte of
		;			BDOS entry point
0000	3A0700		LDA	TOPRAM	;Get MS byte of BDOS entry point
0003	3D		DCR	Α	;Back off one page
0004	2EFF		MVI	L,OFFH	Set LS byte of final address
0006	67		MOV	H, A	HL = XXFFH
0007	F9		SPHL		;Set stack pointer from HL



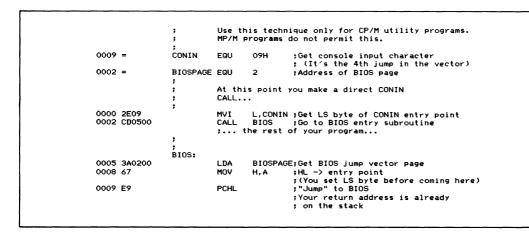


Figure 4-5. Making a direct BIOS call

	;;		This ex ritten t		assumes you have not
0100	,	ORG	100H	:Star	t at TPA
	START:			,	// //
0100 21000	0	LXI	н,о	;Save	2 CCP's stack pointer
0103 39		DAD	SP	;By a	adding it to 0 in HL
0104 220F0	01	SHLD	CCP\$ST	ACK	
0107 31410	01	LXI	SP,LOC	AL\$STA	ск
	;				
	;	The m	ain body	of you	ur program is here
	;				
	;	a	nd when	you are	e ready to return
	;		to the	CCP	i
010A 2A0F0	01	LHLD	CCP\$ST	ACK	;Get CCP's stack pointer
010D F9		SPHL			;Restore SP
010E C9		RET			;Return to the CCP
	;				
010F	CCP\$STAC	ж:	DS	2	;Save area for CCP SP
0111			DS	48	;Local stack
	LOCAL\$S1	ACK:			
0141		END	START		

Figure 4-6. Returning to CCP at program end

Returning to CP/M

Once your program has run, you will need to return control back to CP/M. If your program has not overwritten the CCP and has left the stack pointer as it was when your program was entered, you can return directly to the CCP using a RET instruction.

Figure 4-6 shows how a normal program would do this if you use a local stack, one within the program. The CCP stack is too small; it has room for only 24 16-bit values.

The advantage of returning directly to the CCP is speed. This is true especially on a hard disk system, where the time needed to perform a warm boot is quite noticeable.

If your program has overwritten the CCP, you have no option but to transfer control to location 0000H and let the warm boot occur. To do this, all you need do is execute

EXIT: JMP 0 ;Warm Boot

(As a hint, if you are testing a program and it suddenly exits back to CP/M, the odds are that it has inadvertently blundered to location 0000H and executed a warm boot.)

What the BDOS Does BDOS Function Calls Naming Conventions Making a BDOS Function Request



The Basic Disk Operating System

The Basic Disk Operating System is the real heart of CP/M. Unlike the Console Command Processor, it must be in memory all the time. It provides all of the input/output services to CP/M programs, including the CCP.

As a general rule, unless you are writing a system-dependent utility program, you should use the BDOS for *all* of your program's input/output. If you circumvent the BDOS you will probably create problems for yourself later.

What the BDOS Does

The BDOS does all of the system input/output for you. These services can be grouped into two types of functions:

Simple Byte-by-Byte I/O

This is sending and receiving data between the computer system and its logical devices—the console, the "reader" and "punch" (or their substitutes), and the printer.

Disk File I/O

This covers such tasks as creating new files, deleting old files, opening existing files, and reading and writing 128-byte long "records" to and from these files.

The remainder of this chapter explains each of the BDOS functions, shows how to make each operating system request, and gives additional information for each function. You should also refer to Digital Research's manual, CP/M 2 Interface Guide, for their standard description of these functions.

BDOS Function Calls

The BDOS function calls are described in the order of their function code numbers. Figure 5-1 summarizes these calls.

Naming Conventions

In practice, whenever you write programs that make BDOS calls, you should include a series of equates for the BDOS function code numbers. We shall be making reference to these values in subsequent examples, so they are shown in Figure 5-2 as they will appear in the programs.

The function names used to define the equates in Figure 5-2 are shorter than those in Figure 5-1 to strike a balance between the abbreviated function names used in Digital Research's documentation and the need for clearer function descriptions.

Making a BDOS Function Request

All BDOS functions are requested by issuing a CALL instruction to location 0005H. You can also request a function by transferring control to location 0005H with the return address on the stack.

In order to tell the BDOS what you need it to do, you must arrange for the internal registers of the CPU to contain the required information before the CALL instruction is executed.

٦

Function Code	Description	
	Simple Byte-by-Byte I/O	
0	Overall system and BDOS reset	
1	Read a byte from the console keyboard	
2	Write a byte to the console screen	
3	Read a byte from the logical reader device	
4	Write a byte to the logical punch device	
5	Write a byte to the logical list device	
6	Direct console I/O (no CCP-style editing)	
7*	Read the current setting of the IOBYTE	
8*	Set a new value of the IOBYTE	
9	Send a "\$"-terminated string to the console	
10	Read a string from the console into a buffer	
11	Check if a console key is waiting to be read	
12	Return the CP/M version number	
	Disk File I/O	<i>i</i>
13	Reset disk system	
13	Select specified logical disk drive	
15	Open specified file for reading/writing	
15	Close specified file after reading/writing	
10	Search file directory for first match with filename	
18	Search file directory for next match with filename	
19	Delete (erase) file	
20	Read the next "record" sequentially	
20	Write the next "record" sequentially	
22	Create a new file with the specified name	
23	Rename a file to a new name	
23	Indicate which logical disks are active	
25	Return the current default disk drive number	
25	Set the DMA address (read/write address)	
20 27	Return the address of an allocation vector	
28*	Set specified logical disk drive to Read-Only status	
29	Indicate which disks are currently Read-Only status	
30	Set specified file to System or Read-Only status	
30	Return address of disk parameter block (DPB)	
32*	Set/Get the current user number	
33	Read a "record" randomly	
33	Write a "record" randomly	
35	Return logical file size (even for random files)	
36	Set record number for the next random read/write	
30	Reset specified drive	
40	Write a "record" randomly with zero fill	*These do not work under MP/M.

Figure 5-1. BDOS function calls

0000 =	B\$SYSRESET	EQU	0	;System Reset
0001 =	B\$CONIN	EQU	1	;Read Console Byte
0002 =	B\$CONOUT	EQU	2	;Write Console Byte
0003 =	B\$READIN	EQU	з	;Read "Reader" Byte
0004 =	B\$PUNOUT	EQU	4	;Write "Punch" Byte
0005 =	B\$LISTOUT	EQU	5	;Write Printer Byte
0006 =	B\$DIRCONIO	EQU	6	;Direct Console I/O
0007 =	B\$GETI0	EQU	7	;Get IOBYTE
0008 =	B\$SETI0	EQU	8	;Set IOBYTE
0009 =	B\$PRINTS	EQU	9	Print Console String
000A =	B\$READCONS	EQU	10	Read Console String;
000B =	B\$CONST	EQU	11	;Read Console Status
000C =	B\$GETVER	EQU	12	;Get CP/M Version Number
000D =	B\$DSKRESET	EQU	13	;Disk System Reset
000E =	B\$SELDSK	EQU	14	;Select Disk
000F =	B\$OPEN	EQU	15	;Open File
0010 =	B\$CLOSE	EQU	16	;Close File
0011 =	B\$SEARCHF	EQU	17	;Search for First Name Match
0012 =	B\$SEARCHN	EQU	18	;Search for Next Name Match
0013 =	B\$ERASE	EQU	19	;Erase (delete) File
0014 =	B\$READSEQ	EQU	20	;Read Sequential
0015 =	B\$WRITESEQ	EQU	21	;Write Sequential
0016 =	B\$CREATE	EQU	22	;Create File
0017 =	B\$RENAME	EQU	23	;Rename File
0018 =	B\$GETACTDSK	EQU	24	;Get Active (Logged-in) Disks
0019 =	B\$GETCURDSK	EQU	25	;Get Current Default Disk
001A =	B\$SETDMA	EQU	26	;Set DMA (Read/Write) Address
001B =	B\$GETALVEC	EQU	27	;Get Allocation Vector Address
001C =	B\$SETDSKR0	EQU	28	;Set Disk to Read Only
001D =	B\$GETRODSKS	EQU	29	;Get Read Only Disks
001E =	B\$SETFAT	EQU	30	;Set File Attributes
001F =	B\$GETDPB	EQU	31	;Get Disk Parameter Block Address
0020 =	B\$SETGETUN	EQU	32	;Set/Get User Number
0021 =	B\$READRAN	EQU	33	Read Random
0022 =	B\$WRITERAN	EQU	34	;Write Random
0023 =	B\$GETFSIZ	EQU	35	;Get File Size
0024 =	B\$SETRANREC	EQU	36	Set Random Record Number
0025 =	B\$RESETD	EQU	37	Reset Drive
0028 =	B\$WRITERANZ	EQU	40	Write Random with Zero-Fill

Figure 5-2. Equates for BDOS function code numbers

The function code number of the specific function call you want performed must be in register C.

If you need to hand a single-byte value to the BDOS, such as a character to be sent to the console, then you must arrange for this value to be in register E. If the value you wish to pass to the BDOS is a 16-bit value, such as the address of a buffer or a file control block (FCB), this value must be in register pair DE.

When the BDOS hands back a single-byte value, such as a keyboard character or a return code indicating the success or failure of the function you requested, it will be returned in register A. When the BDOS returns a 16-bit value, it will be in register pair HL.

On return from the BDOS, registers A and L will contain the same value, as will registers B and H. This odd convention stems from CP/M's origins in PL/M (Programming Language/Microprocessor), a language used by Intel on their MDS system. Thus, PL/M laid the foundations for what are known as "register calling conventions."

The BDOS makes no guarantee about the contents of the other registers. If you need to preserve a value that is in a register, either store the value in memory or push it onto the stack. The BDOS uses its own stack space, so there is no need to worry about it consuming your stack.

To sum up, when you make a function request to the BDOS that requires a byte value, the code and the required entry and exit parameters will be as follows:

MVI	C, FUNCTION\$CODE	<pre>;C = function code</pre>
MVI	E,SINGLE\$BYTE	;E = single byte value
CALL	BDOS	;Location 5
		;A = return code or value
		;or HL = return value

For those function requests that need to have an address passed to the BDOS, the calling sequence is

MVI LXI CALL	C,FUNCTION\$CODE · D,ADDRESS BDOS	;C = function code ;DE = address ;Location 5 :A = return code or value
		;A = return code or value ;or HL = return value

If a function request involves disk files, you will have to tell the BDOS the address of the FCB that you have created for the file. (Refer back to Chapter 3 for descriptions of the FCB.)

Many file processing functions return a value in register A that is either 0FFH, indicating that the file named in the FCB could not be found, or equal to a value of 0, 1, 2, or 3. In the latter case, the BDOS is returning what is called a "directory code." The number is the directory entry number that the BDOS matched to the file name in your FCB. At any given moment, the BDOS has a 128-byte sector from the directory in memory. Each file directory entry is 32 bytes, so four of them (numbered 0, 1, 2, and 3) can be processed at a time. The directory code indicates which one has been matched to your FCB.

References to CP/M "records" in the following descriptions mean 128-byte sectors. Do not confuse them with the logical records used by applications programs. Think of CP/M records as 128-byte sectors throughout.

Function 0: System Reset

Function Code: C = 00HEntry Parameters: None Exit Parameters: Does not return

Example

0000 =	B\$SYSRESET	EQU	0	;System Reset
0005 =	BDOS	EQU	5	;BDOS entry point
0000 0E00 0002 C30500	MVI JMP	C,B\$SYS BDOS	RESET	;Set function code ;Note: you can use a JMP since ; you don't get control back

Purpose The system reset function makes CP/M do a complete reset, exactly the same as the warm boot function invoked when you transfer control to the WARM-BOOT point (refer to Figure 4-1).

In addition to resetting the BDOS, this function reloads the CCP, rebuilds the allocation vectors for the currently logged disks, sets the DMA address (used by CP/M to address the disk read/write buffer) to 80H, marks all disks as being Read/Write status, and transfers control to the CCP. The CCP then outputs its prompt to the console.

Notes This function is most useful when you are working in a high-level language that does not permit a jump instruction to an absolute address in memory. Use it when your program has finished and you need to return control back to CP/M.

Function 1: Read Console Byte

Function Code: C = 01 HEntry Parameters: None Exit Parameters: A = Data byte from console

Example

0001 =	B\$CONIN	EQU 1	;Console input
0005 =	BDOS	EQU 5	;BDOS entry
0000 0E01	MVI	C,B\$CONIN	;Get function code
0002 CD0500	CALL	BDOS	

Purpose

This function reads the next byte of data from the console keyboard and puts it into register A. If the character input is a graphic character, it will be echoed back to the console. The only control characters that are echoed are CARRIAGE RETURN, LINE FEED, BACKSPACE, and TAB. In the case of a TAB character, the BDOS outputs as many spaces as are required to move the cursor to the next multiple of eight columns. All of the other control characters, including CONTROL-C, are input but are not echoed.

This function also checks for CONTROL-S (XOFF) to see if console output should be suspended, and for CONTROL-P (printer echo toggle) to see if console output should also be sent to the list device. If CONTROL-S is found, further output will be suspended until you type another character. CONTROL-P will enable the echoing of console output the first time it is pressed and disable it the second time.

If there is no incoming data character, this function will wait until there is one.

Notes This function often hinders rather than helps, because it echoes the input. Whenever you need console input at the byte-by-byte level, you will usually want to suppress this echo back to the console. For instance, you may know that the "console" is actually a communications line such as a modem. You may be trying to accept a password that should not be echoed back. Or you may need to read a cursor control character that would cause an undesirable side effect on the terminal if echoed there.

In addition, if you need more than a single character from the console, your program will be easier to use if the person at the console can take full advantage of the CCP-style line editing. This can best be done by using the Read Console String function (code 10, 0AH).

Read Console String also is more useful for single character input, especially when you are expecting a "Y" or "N" (yes or no) response. If you use the Read Console Byte function, the operator will have only one chance to enter the data. When you use Read Console String, however, users have the chance to type one character, change their minds, backspace, and type another character.

Function 2: Write Console Byte

Function Code: C = 02HEntry Parameters: E = Data byte to be output Exit Parameters: None

Example

0002		B\$CONOUT	EQU	2	;Write Console Byte
0005		BDOS	EQU	5	;BDOS entry
0002	0E02 1E2A CD0500	MVI MVI CALL	C,B\$CON E,'*' BDOS	DUT	;Function code ;E = data byte to be output

- **Purpose** This function outputs the data byte in register E to the console. As with function 1, if the data byte is a TAB character, it will be expanded by the BDOS to the next column that is a multiple of eight. The BDOS also checks to see if there is an incoming character, and if there is, checks to see if it is a CONTROL-S (in which case console output is suspended) or CONTROL-P (in which case echoing of console output to the printer is toggled on or off).
- **Notes** You may have problems using this function to output cursor-addressing control sequences to the console. If you try to output a true binary cursor address to position 9, the BDOS will interpret this as a TAB character (ASCII code 9) and dutifully replace it with zero to eight blanks. If you need to output binary values, you must set the most significant bit of the character (use an ORI 80H, for example) so that it will not be taken as the ASCII TAB.

Here are two general-purpose subroutines that you will need for outputting messages. The first one, shown in Figure 5-3, outputs a null-byte-terminated message from a specified address. The second, in Figure 5-4, does essentially the same thing *except* that the message string follows immediately after the call to the subroutine.

	;MSGOUT (mess ;Output null-		ninated	message.
	;Calling sequ	ience		
	; MESS/	AGE:	DB	'Message',0
	;	:		
	; LXI	H, MESS		
	; CALL	MSGOUT	r	
	;Exit Paramet	ters		
	; HL -:	Null by	te termi	nator
0002 =	B\$CONOUT	EQU	2	;Write Console Byte
0005 =	BDOS	EQU	5	;BDOS entry point
	MSGOUT:			
0000 7E	MOV	Α,Μ		;Get next byte for output
0001 B7	ORA	A		
0002 C8	RZ			;Return when null-byte
0003 23	INX	н		;Update message pointer
0004 E5	PUSH	н		Save updated pointer;
0005 5F	MOV	E,A		Ready for BDOS
0006 OE02	MVI	C,B\$C(DNOUT	
0008 CD0500	CALL	BDOS		
000B E1	POP	н		Recover message pointer;
0000 030000	JMP	MSGOUT	r	;Go back for next character

Figure 5-3. Write console byte example, output null-byte terminated message from specified address

```
;MSGOUTI (message out in-line)
;Output null-byte-terminated message that
;follows the CALL to MSGOUTI.
                   ;Calling sequence
                             CALL
                                       MSGOUTI
                   :
                             DB
                                       'Message',0
                   :
                             ... next instruction
                   ;
                   ;Exit Parameters
                            HL -> instruction following message
                   ;
0002 =
                   B$CONOUT
                                       EQU
                                                 25
                                                           ;Write Console Byte
;BDOS entry point
0005 =
                   BDOS
                                       EQU
                   MSGOUTI:
0000 E1
                             POP
                                       н
                                                           ;HL ~> message
0001 7E
0002 23
                             MOV
                                       A,M
                                                           ;Get next data byte
                             INX
                                       H
                                                           ;Update message pointer
0003 B7
                             ORA
                                       Α
                                                           ;Check if null byte
0004 C20800
                             JNZ
                                       MSGOUTIC
                                                           ;No, continue
                             PCHL
                                                           ;Yes, return to next instruction
; after in-line message
0007 E9
                  MSGOUTIC:
                             PUSH
0008 E5
                                                           ;Save message pointer
;Ready for BDOS
                                       н
0009 5F
                                       E,A
                             MOV
000A 0E02
000C CD0500
                             MVI
                                       C, B$CONOUT
                                                           ;Function code
                             CALL
                                       BDOS
000F C30000
                             JMP
                                       MSGOUTI
                                                           ;Go back for next char.
```

Figure 5-4. Write console byte example, output null-byte terminated message following call to subroutine

Function 3: Read "Reader" Byte

Function Code:	C = 03H
Entry Parameters:	None
Exit Parameters:	A = Character input

Example

Notes

0003 =	B\$READIN	EQU 3	;Read "Reader" Byte
0005 =	BDOS	EQU 5	;BDOS entry
0000 0E03	MVI	C,B\$READIN	;Function code
0002 CD0500	CALL	BDOS	;A = reader byte

Purpose This function reads the next character from the logical "reader" device into register A. In practice, the physical device that is accessed depends entirely on how your BIOS is configured. In some systems, there is no reader at all; this function will return some arbitrary value such as 1AH (the ASCII CONTROL-Z character, used by CP/M to denote "End of File").

Control is not returned to the calling program until a character has been read.

Since the physical device (if any) used when you issue this request depends entirely on your particular BIOS, there can be no default standard for all CP/M implementations. This is one of the weaker parts of the BDOS.

You should "connect" the reader device by means of BIOS software to a serial port that can be used for communication with another system. This is only a partial solution to the problem, however, because this function call does not return control to your program until an incoming character has been received. There is no direct way that you can "poll" the reader device to see if an incoming character has been received. Once you make this function call, you lose control until the next character arrives; there is no function corresponding to the Read Console Status (function code 11, 0BH) that will simply read status and return to your program.

One possible solution is to build a timer into the BIOS reader driver that returns control to your program with a dummy value in A if a specified period of time goes by with no incoming character. But this brings up the problem of what dummy value to use. If you ever intend to send and receive files containing pure binary information, there is no character in ASCII that you might not encounter in a legitimate context. Therefore, any dummy character you might choose could also be true data.

The most cunning solution is to arrange for one setting of the IOBYTE (which controls logical-device-to-physical-device mapping) to connect the console to the serial communication line. This done, you can make use of the Read Console Status function, which will return not the physical console status but the serial line status. Your program can then act appropriately if no characters are received within a specified time. Figure 5-11 shows a subroutine that uses this technique in the Set IOBYTE function (code 8, 08H).

76 The CP/M Programmer's Handbook

Figure 5-5 shows an example subroutine to read lines of data from the reader device. It reads characters from the reader, stacking them in memory until either a LINE FEED or a specified number of characters has been received. Note that CARRIAGE RETURNS are ignored, and the input line is terminated by a byte of 00H. The convention of 00H-byte terminated strings and no CARRIAGE RETURNS is used because it makes for much easier program logic. It also conforms to the conventions of the C language.

	;Carriage ;when spec		ignored. r of chai	, and input terminates acters have been read
	;timeout i	n this subr	outine.	that there is no It will wait forever at the reader device.
	;Calling s ; LX ; LX ; CA	I H, BUF	COUNT	
	; BC	-> 00H byt	count (ating string D if max. chars.read) ad
0003 = 0005 =	B\$READIN BDOS	EQU EQU	3 5	;Reader input ;BDOS entry point
= 0000 = A000	CR LF	EQU EQU	ODH OAH	;Carriage return ;Line feed (terminator)
0000 79 0001 B0 0002 5F 0003 CA2000 0006 C5 0007 E5		A B V E,A	RX	;Check if count 0 ;If count 0 on entry, fake ; last char. read (00H) ;Yes, exit ;Save max. chars. count ;Save buffer pointer
0008 0E03 000A CD0500 000D 5F 0010 CA0800 0013 E1 0014 C1 0015 FE0A 0017 CA2000 001A 77 001B 23 001C 0B 001D C30000	RL\$RDRI: MV CA MG CF JZ PC CF JZ CF JZ MC IN DC	LL BDOS V E,A I CR RL\$RI P H P B I LF RL\$RI V M,A IX H X B	RX	;Loop back here to ignore ;A = character input ;Preserve copy of chars. ;Check if carriage return ;Yes, ignore it ;Recover buffer pointer ;Recover max. Count ;Check if line feed ;Yes, exit ;No, store char. in buffer ;Update buffer pointer ;Downdate count ;Loop back for next char.
0020 3600 0022 C9	RL\$RDRX: MV RE			;Null-byte-terminate buffer

Function 4: Write "Punch" Byte

Function Code: C = 04HEntry Parameters: E = Byte to be output Exit Parameters: None

Example

0004 =	B\$PUNOUT	EQU 4	;Write "Punch" Byte
0005 =	BDOS	EQU 5	
0000 0E04 0002 1E2A 0004 CD0500	MVI MVI CALL	C,B\$PUNOUT E,'*' BDOS	;Function code ;Data byte to output

- **Purpose** This function is a counterpart to the Read "Reader" Byte described above. It outputs the specified character from register E to the logical punch device. Again, the actual physical device used, if any, is determined by the BIOS. There is no set standard for this device; in some systems the punch device is a "bit bucket," so called because it absorbs all data that you output to it.
- **Notes** The problems and possible solutions discussed under the Read "Reader" Byte function call also apply here. One difference, of course, is that this function outputs data, so the problem of an indefinite loop waiting for the next character is less likely to occur. However, if your punch device is connected to a communications line, and if the output hardware is not ready, the BIOS line driver will wait forever. Unfortunately, there is no legitimate way to deal with this problem since the BDOS does not have a function call that checks whether a logical device is ready for output.

Figure 5-6 shows a useful subroutine that outputs a 00H-byte terminated string to the punch. Wherever it encounters a LINE FEED, it inserts a CARRIAGE RETURN into the output data.

Function 5: Write List Byte

Function Code: C = 05HEntry Parameters: E = Byte to be output Exit Parameters: None

Example

0005 =	B\$LSTOUT	EQU 5	;Write List Byte
0005 =	BDOS	EQU 5	
0000 0E05 0002 1E2A 0004 CD0500	MVI MVI CALL	C,B\$LSTOUT E,'*' BDOS	;Function code ;Data byte to output

Purpose This function outputs the specified byte in register E to the logical list device. As with the reader and the punch, the physical device used depends entirely on the BIOS.

;WL\$PUN ;Write line to punch device. Output terminates ;when a OOH byte is encountered. ;A carriage return is output when a line feed is ;encountered. ;Calling sequence : LXI H H.BUFFER : CALL WL\$PUN :Exit parameters HL -> OOH byte terminator ٠ 0004 = **B\$**PUNOUT EQU 0005 = BDOS EQU 5 000D = CR EOU ODH :Carriage return 000A = 1 F EQU OAH ;Line feed WL\$PUN: PUSH ;Save buffer pointer 0000 E5 н ;Get next character ;Check if 00H MOV A. M 0001 7E 0002 B7 ORA WL\$PUNX 0003 CA2000 JZ ;Yes, exit CPI 0006 FE0A LF ;Check if line feed 0008 CC1600 cΖ WL\$PUNLF ;Yes, O/P CR 000B 5F MOV E,A ;Character to be output C, B\$PUNOUT 000C 0E04 MVI ;Function code 000F CD0500 CALL BDOS ;Output character 0011 E1 POP H, Recover buffer pointer :Increment to next char. 0012 23 INX н 0013 C30000 WL\$PUN ;Output next char . IMP WL\$PUNLF: :Line feed encountered C, B\$PUNOUT 0016 0E04 MVI ;Function code 0018 1E0D MVI E,CR ;Output a CR 001A CD0500 CALL BDOS 001D 3EOA MVI ;Recreate line feed A.LF 001F C9 RET ;Output LF WL \$PUNX: :Exit POP ;Balance the stack 0020 E1 н 0021 C9 RET

Figure 5-6. Write line to punch device

Notes

One of the major problems associated with this function is that it does not deal with error conditions very intelligently. You cannot be sure which physical device will be used as the logical list device, and most standard BIOS implementations will cause your program to wait forever if the printer is not ready or has run out of paper. The BDOS has no provision to return any kind of error status to indicate that there is a problem with the list device. Therefore, the BIOS will have to be changed in order to handle this situation.

Figure 5-7 is a subroutine which outputs data to the list device. As you can see, this is essentially a repeat of Figure 5-6, which performs the same function for the logical punch device.

1						
		;when a	OOH by lage re	te is en	ncountere	utput terminates ed. when a line feed is
		;Calling	a seque	nce		
			LXI CALL	H, BUFI WL\$LS		
		;Exit pa ;			e termina	ator
	0005 =	B\$LSTOUT	r	EQU	5	
	0005 =	BDOS		EQU	5	
	000D =	CR		EQU	ODH	;Carriage return
	000A =	LF		EQU	OAH	;Line feed
		WL\$LST:				
	0000 E5		PUSH	н		;Save buffer pointer
	0001 7E		MOV	Α,Μ		;Get next character
	0002 B7		ORA	A		;Check if OOH
	0003 CA2000		JZ	WL\$LS	ТХ	;Yes, exit
	0006 FEOA		CPI	LF		;Check if line feed
	0008 CC1600		CZ	WL\$LS	TLF	;Yes, O/P CR
	000B 5F		MOV	E,A		;Character to be output
	000C 0E05		MVI	C,B\$LS	51001	;Function code
	000E CD0500		CALL	BDOS		;Output character
	0011 E1		POP	н		Recover buffer pointer
}	0012 23 0013 C30000		INX JMP	H WL\$LS	-	Update to next char.
	0013 030000		UNF	WLPLO	,	;Output next char.
		WL\$LSTLF	· .			;Line feed encountered
	0016 0E05		MVI	C,B\$L	STOUT	;Function code
	0018 1EOD		MVI	E,CR		;Output a CR
1	001A CD0500		CALL	BDOS		
1	001D 3EOA		MVI	A,LF		;Recreate line feed
}	001F C9		RET			;Output LF
		WL\$LSTX:				;Exit
	0020 E1		POP	н		Balance the stack
1	0021 C9		RET			

Figure 5-7. Write line to list device

Function 6: Direct Console I/O

Function Code:	C = 06H
Entry Parameters:	E = 0FFH for Input
	E = Other than 0FFH for output
Exit Parameters:	A = Input byte or status

Example

0006 = 0005 =	B\$DIRCONIO BDOS	EQU 6 EQU 5	;Direct (raw) Console I/O ;BDOS entry point
			;Example of console input
0000 0E06 0002 1EFF 0004 CD0500	MVI MVI CALL	C,B\$DIRCON E,OFFH BDOS	IO ;Function code ;OFFH means input ;A = 00 if no char. waiting ;A = NZ if character input

			,Example of console output
0007 0E06 0009 1E2A 000B CD0500	MVI MVI CALL	C,B\$DIRCONIO E,'*' BDOS	;Function code ;Not OFFH means output char.

•Example of console output

Purpose This function serves double duty: it both inputs and outputs characters from the console. However, it bypasses the normal control characters and line editing features (such as CONTROL-P and CONTROL-S) normally associated with console I/O. Hence the name "direct" (or "unadorned" as Digital Research describes it). If the value in register E is *not* 0FFH, then E contains a valid ASCII character that is output to the console. The logic used is most easily understood when written in pseudo-code:

```
if this is an input request (E = OFFH)
{
    if console status indicates a character is waiting
        {
            read the char from the console and
            return to caller with char in A
        }
    else (no input character waiting) and
            return to caller with A = 00
    }
else (output request)
    {
        output the char in E to the console and
        return to caller
    }
}
```

Notes

This function works well provided you never have to send a value of 0FFH or expect to receive a value of 00H. If you do need to send or receive pure binary data, you cannot use this function, since these values are likely to be part of the data stream.

To understand why you might want to send and receive binary data, remember that the logical "reader" does not have any method for you to check its status to see if an incoming character has arrived. All you can do is attempt to read a character (Read Reader Byte, function code 3). However, the BDOS will not give control back to you until a character arrives (which could be a very long time). One possibility is to logically assign the console to a communications line by the use of the IOBYTE (or some similar means) and then use this Direct I/O call to send and receive data to and from the line. Then you could indeed "poll" the communications line and avoid having your program go into an indefinite wait for an incoming character. An example subroutine using this technique is shown in Figure 5-11 under Set IOBYTE (function code 8).

Figure 5-8 shows a subroutine that uses the Direct Console Input and Output. Because this example is more complex than any shown so far, the code used to check the subroutine has also been included.

Function 7: Get IOBYTE Setting

Function Code: C = 07HEntry Parameters: None Exit Parameters: A = IOBYTE current value

TESTRED CODE ;Because of the complexity of this subroutine, the ; actual testbed code has been left in this example. ; It assumes that DDT or ZSID ; will be used for checkout. IF ;Change to IF 0 to disable testbed 1 0100 ORG 100H 0100 C31101 START JMP ;Bypass "variables" setup by DDT 0103 00 OPTIONS: DB ;Option flags 0104 41454900 'A','E','I',0 TERMS: **DB** ;Terminators 0108 05 BUFFFR 5 **DB** ;Max. characters in buffer ;Actual count 0109 00 DB ő 010A 6363636363 DB 99,99,99,99,99 ;Data bytes 010F 6363 DB 99.99 START: 0111 210801 LXI H, BUFFER ;Get address of buffer 0114 110401 0117 3A0301 LXI D, TERMS ;Address of terminator table OPTIONS ;Get options set by DDT I DA 011A 47 011B CD2B01 MOV B, A ;Put in correct register CALL RCS ;Enter subroutine ;Force DDT breakpoint 011E CD3800 CALL 38H 0121 C31101 START ;Test again ;End of testbed . IMP ENDIF ;RCS: Read console string (using raw input) ;Reads a string of characters into a memory ; buffer using raw input. :Supports options: o to echo characters or not (when echoing, : a carriage return will be echoed followed . by line feed) o warm boot on input of control-C or not . : o terminating input either on: o max. no of chars input : o matching terminator character : Calling Sequence ; LXI H, BUFFER Buffer has structure: BUFFER: DB 10 Max. size . **DB** 0 Actual Read 10+1 DS Buffer area . MVI B. OPTIONS Options required . (see equates) : LXI D, TERMS Pointer to OOH-byte : terminated Chars, any one of which is a terminator. . CALL RCS ; Exit Parameters : BUFFER: Updated with data bytes and actual : character count input. 2 (Does not include the terminator). 3 A = Terminating Code ; 0 = Maximum number of characters input. : NZ = Terminator character found. 0001 = RCS\$ECH0 EQU 0000\$0001B ;Input characters to be echoed 0002 ≠ RCS\$ABORT EQU 0000\$0010B ;Abort on Control-C 0004 =RCS\$FOLD FOU 0000\$0100B ;Fold lowercase to uppercase ;DE -> term. char. set 0008 = RCS\$TERM FOIL 0000\$1000B 0006 = B\$DIRCONIO EQU ;Direct console I/O 6 0005 = 5 BDOS EQU ;BDOS entry point 0003 = CTL\$C EQU озн ;Control-C 000D = CR EQU орн ;Carriage return

Figure 5-8. Read/write string from/to console using raw I/O

82

		· · · · · · · · · · · · · · · · · · ·					
	000A	=	1 F		EQU	OAH	;Line feed
	0008		BS		EQU	08H	;Backspace
	0124	0.0	RCS\$ST:	DB	ODH		;Internal standard terminator table ;Carriage return
	0125			DB	OAH		;Line feed
	0126			DB	õ		;End of table
	0107	08200800	RCS\$BSS	DB	BS, ′ ′,	BC O	;Destructive backspace sequence
	012/	08200800		00	DO, ,	50,0	
			RCS:				;<<<< Main entry
	012B			INX	н		;HL -> actual count
	012C 012E			MVI DCX	м,о н		;Reset to initial state ;HL -> max, count
	VILL	20		DOX			yne y mant coant
			RCS\$L:				
	012F			PUSH	H RCS\$GC		;Save buffer pointer
	0130	CD9201		CALL	RU3⊅00		;Get character and execute: ; ECHO, ABORT, and FOLD options
							;C = character input
	0133			POP	н		Recover buffer pointer;
	0134 0136	3E08		MVI ANA	A, RCS\$1 B	ERM	;Check if user-specified terminator ;B = options
	0137	A0 C23D01		JNZ	RCS\$US1	r	;User specified terminators
		112401		LXI	D,RCS\$S	ST	;Standard terminators
			DCCAUCT				
	0130	CDD401	RCS\$UST	CALL	RCS\$CT		;Check for terminator
	0140	CA4C01		JZ	RCS\$NO1	гт	Not terminator
	0143			MOV	B,A		;Preserve terminating char.
			RCS\$MCI				;(Max. char. input shares this code)
	0144	0E00	RU3⊅MU1	MVI	с,о		;Terminate buffer
	0146	CD7F01		CALL	RCS\$SC		;Save character
	0149			MOV	Α,Β		Recover terminating char.
	014A 014B			'ORA RET	A		;Set flags
	0145	07					
			RCS\$NOT				;Not a terminator
	014C 014E	3E08		MVI CMP	A,BS C		;Check för backspace
		CA6001		JZ	RCS\$BS		;Backspace entered
		CD7F01		CALL	RCS\$SC		;Save character in buffer
	0155	CD8B01		CALL	RCS\$UC		;Update count
	0158	C22F01		JNZ MVI	RCS∳L B.O		;Not max. so get another char. ;Fake terminating char.
		0600 C34401		JMP	RCS\$MC1	[;A = 0 for max. chars. input
			RCS\$BS:	-			;Backspace entered
	0160 0161			PUSH INX	н н		;Save buffer pointer ;HL -> actual count
	0162			DCR	M		;Back up one
	0163	FA7A01		JM	RCS\$NBS		;Check if count negative
	0166	212701 3E01		LXI MVI	H,RCS\$E A,RCS\$E		;HL -> backspacing sequence ;No, check if echoing
	0169 016B			ANA	A, KUS⊅t B	Joho	;BS will have been echoed if so
	016C	CA7001		JZ	RCS\$BSN	ŧΕ	;No, input BS not echoed
	016F	23		INX	н		;Bypass initial backspace
			RCS\$BSN	E:			
	0170			PUSH	в		;Save options and character
	0171			PUSH	D		;Save terminator table pointer
	0172	CDF601		CALL POP	WCS D		;Write console string ;Recover terminator table pointer
	0176			POP	В		Recover options and character
		C37B01		JMP	RCS\$BS)	(;Exit from backspace logic
			DOCANDO	_			
	017A	34	RCS\$NBS	INR	м		;Reset count tò 0
ĺ	V1/M	~ 1					,
1			RCS\$BSX				· De la companya de la
	017B	E1 C32F01		POP JMP	H RCS\$L		;Recover buffer pointer ;Get next character
l	01/0	0.52001		010	NCJ₽L		yeet ment character
L							

Figure 5-8. (Continued)

		RCS\$SC:			;Save character in C in buffer
				_	;HL -> buffer pointer
017F			PUSH	D	;Save terminator table pointer
0180			PUSH	н	;Save buffer pointer
0181			INX	н	;HL -> actual count in buffer
0182			MOV	E,M	;Get actual count
0183			INR	E	;Count of O points to first data byte
	1600		MVI	D, O	Make word value of actual count;
0186			DAD	D	;HL -> next free data byte
0187			MOV	M,C	;Save data byte away
0188			POP	н	;Recover buffer pointer
0189	D1		POP	D	Recover terminator table;
					; pointer
018A	C9		RET		
		RCS\$UC:			;Update buffer count and check for max.
					;Return Z set if = to max., NZ
					; if not HL -> buffer on entry
018B	E5		PUSH	н	;Save buffer pointer
0180			MOV	A, M	;Get max. count
0180			INX	н	;HL -> actual count
018E			INR	M	;Increase actual count
018F	BE		CMP	M	;Compare max. to actual
0190	FI		POP	н	;Recover buffer pointer
0191			RET		;Z-flag set
~	<u>.</u> ,		· • • • •		,
		RCS\$GC:			;Get character and execute
					; ECHO, ABORT and FOLD options
0192			PUSH	D	;Save terminator table pointer
0193			PUSH	н	;Save buffer pointer
0194	C5		PUSH	В	;Save option flags
		RCS\$WT:			
0195	0E06	1009911	MVI	C, B\$DIRCONIO	;Function code
0197	1EFF		MVI	E, OFFH	;Specify input
	CD0500		CALL	BDOS	Personal Panetas
0190			ORA	A	;Check if data waiting
	CA9501		JZ	RCS\$WT	;Go back and wait
01A0			POP	B	Recover option flags
01A1			MOV	Č,A	;Save data byte
	3E02		MVI	A, RCS\$ABORT	;Check if abort option enabled
01A4			ANA	B	
	CAAE01		JZ	RCS\$NA	:No abort
	3E03		MVI	A, CTL\$C	Check for control-C
01AA			CMP	C	,
	CA0000		JZ	õ	;Warm boot
		Decent			
01AF	3E04	RCS\$NA:	MVI	A, RCS\$FOLD	;Check if folding enabled
01B0			ANA	B	Action II Lording Engiled
	C4E501		CNZ	TOUPPER	;Convert to uppercase
	3E01		MVI	A, RCS\$ECHO	Check if echo required
01B6			ANA	B	Joneck II ECHO IEMUIIEU
	CADIOI		JZ	RCS\$NE	;No echo required
01BA			PUSH	B	;No echo required ;Save options and character
OIBB	59		MOV	E,C	;Move character for output
OIRC	0E06		MVI	C,B\$DIRCONIO	Function code
	CD0500		CALL	BDOS	;Echo character
01C1			POP	BDUS	;Ecno character ;Recover options and character
	3EOD		MVI	A, CR	;Recover options and character ;Check if carriage return
01C4			CMP	A, UK C	Joneck II Carriage return
					•No
0108	C2D101			RCS\$NE	;No
			PUSH	B	Save options and character
01CP	0E06 1E0A		MVI	C,B\$DIRCONIO E.LF	;Function code ;Output line feed
				· · ·	ywwyrdd Adlin Innu
0100	CD0500		CALL POP	BDOS B	Recover options and character
0100	01		106	U	A secover options and character
		RCS\$NE:			-
	E1		POP	н	Recover buffer pointer
			POP	D	Recover terminator table;
01D2 01D3	01		RET		;Character in C

Figure 5-8. (Continued)

	RCS\$CT:		;Check for terminator
			;C = character just input
			;DE -> 00-byte character
			; string of term. chars.
			Returns Z status if no
			; match found, NZ if found
			; (with A = C = terminating
			; character)
	D 116		
01D4 D5	PUS	SH D	;Save table pointer
	RCS\$CTL:		
01D5 1A	LDA	AX D	;Get next terminator character
01D6 B7	OR/		;Check for end of table
01D7 CAE201 01DA B9	JZ	RCS\$CTX	;No terminator matched
01DA B9	CM	• C	Compare to input character
01DB CAE201	JZ	RCS\$CTX	;Terminator matched
01DE 13	IN		:Move to next terminator
01DF C3D501	JMF	P RCS\$CTL	; loop to try next character in table
	DCO#CTV-		·Chark Asumissian suit
150 B7	RCS\$CTX: OR/	A A	Check terminator exit
01E2 B7	UR	, n	At this point, A will either
			; be 0 if the end of the
			; table has been reached, or
			; NZ if a match has been
			; found. The Z-flag will be
			; set.
D1E3 D1	POF		Recover table pointer
01E4 C9	RE	1	
	;TOUPPER ~	Fold lowercase 1	letters to upper
		= Character on er	
	TOUPPER:		
01E5 3E60	MV		;Check if folding needed
D1E7 B9	CM		;Compare to input char.
01E8 D2F501	JNG		;No, char. is $\langle \text{ or } = "a"-1$
1EB 3E7A	MV:	I A, 'z'	;Maybe, char. is = or > "a"
01ED B9	CMI	» с	
DIEE DAF501	JC	TOUPX	;No, char. is > "z"
D1F1 3EDF	MV		;Fold character
01F3 A1	AN		
	MO		Return folded character
1F4 4F			
01F4 4F	TOUDY		
91F4 4F	TOUPX:	۲.	
01F4 4F	RE		
01F4 4F	RE ;WCS - Wri	te console string	; (using raw I/O)
01F4 4F 01F5 C9	RE ;WCS - Wri ;Output ter	te console string rminates when a C	OOH byte is encountered.
01F4 4F	RE ;WCS - Wri ;Output ten ;A carriago	te console string rminates when a C e return is outpu) (using raw I/O) DOH byte is encountered. At when a line feed is
01F4 4F	RE ;WCS - Wri ;Output ter	te console string rminates when a C e return is outpu	OOH byte is encountered.
01F4 4F	RE ;WCS - Wri ;Output ter ;A carriage ;encountere ;Calling se	te console string rminates when a C e return is outpu ed. equence	OOH byte is encountered.
01F4 4F	RE ;WCS - Wri ;Dutput te; ;A carriag; ;encounterc ;Calling s; ; LX	te console string minates when a C e return is outpu ed. equence I H, BUFFER	OOH byte is encountered.
D1F4 4F	RE ;WCS - Wri ;Output ter ;A carriage ;encountere ;Calling s	te console string minates when a C e return is outpu ed. equence I H, BUFFER	OOH byte is encountered.
01F4 4F	RE ;WCS - Wri ;Output ter ;A carriag; ;encounter ;Calling so ; LX ; CAL	te console string minates when a C e return is outpu ed. equence I H,BUFFER _L WCS	OOH byte is encountered.
01F4 4F	RE ;WCS - Wri ;Output ter ;A carriag ;encounterd ;Calling sd ; LX ; CAl ;Exit param	te console string minates when a C e return is outpu ed. equence I H,BUFFER _L WCS	00H byte is encountered. it when a line feed is
01F4 4F	RE ;WCS - Wri ;Output te ;A carriage ;encounter ;Calling s, ; LX ; CAL ;Exit paran ; HL	te console string minates when a C e return is outpu ed. equence I H,BUFFER L WCS neters	00H byte is encountered. it when a line feed is
D1F4 4F	RE ;WCS - Wri ;Output ter ;A carriag; ;encounterc ;Calling sc ; LX ; CAL ;Exit paran ; HL WCS:	te console string minates when a C e return is outpu ed. equence I H,BUFFER L WCS meters -> 00H byte term	NOH byte is encountered. It when a line feed is
D1F4 4F D1F5 C9 D1F6 E5	RE ;WCS - Wri ;Output te ;A carriag; ;encounterd ;Calling sc ;Calling sc ;Call	te console string minates when a C e return is outpu equence I H,BUFFER _L WCS meters -> OOH byte term 5H H	<pre>DOH byte is encountered. ut when a line feed is ninator ;Save buffer pointer</pre>
01F4 4F 01F5 C9 01F6 E5 01F6 77E	RE ;WCS - Wri ;Output te ;A carriage ;encounter ;Calling s. ; LX ; CAL ;Exit param ; HL WCS: PU: MO	te console string minates when a C e return is outpu ed. equence I H,BUFFER _L WCS neters _> OOH byte term 5H H V A,M	NOH byte is encountered. ut when a line feed is ninator ;Save buffer pointer ;Get next character
01F4 4F 01F5 C9 01F6 E5 01F6 77E	RE ;WCS - Wri ;Output te ;A carriag; ;encounterd ;Calling sc ;Calling sc ;Call	te console string minates when a C ereturn is outpu equence I H,BUFFER L WCS meters -> 00H byte term SH H A A,M A A	00H byte is encountered. It when a line feed is ninator ;Save buffer pointer ;Get next character ;Check if 00H
D1F4 4F D1F5 C9 D1F6 E5 D1F7 7E D1F7 7E D1F8 B7	RE ;WCS - Wri ;Output te ;A carriage ;encounter ;Calling s. ; LX ; CAL ;Exit param ; HL WCS: PU: MO	te console string minates when a C e return is outpu equence I H,BUFFER L WCS meters -> 00H byte term SH H V A,M A A	00H byte is encountered. It when a line feed is ninator ;Save buffer pointer ;Get next character ;Check if 00H
D1F4 4F D1F5 C9 D1F5 E5 D1F6 E5 D1F7 7E D1F8 B7 D1F9 CA1602	RE ;WCS - Wri ;Output ter ;A carriag ;encounter ;Calling s ;Calling s ; LX ; LX ; LX ; CAL ;Exit para ; HL WCS: PUS MO OR	te console string minates when a C ereturn is outpu ed. equence I H,BUFFER _L WCS neters -> OOH byte term SH H V A,M A A WCSX	NOH byte is encountered. ut when a line feed is ninator ;Save buffer pointer ;Get next character
01F4 4F 01F5 C9 01F6 E3 01F7 7E 01F8 B7 01F9 CA1602 01FC E0A	RE ;WCS - Wri ;Output te ;A carriage ;encounter ;Calling s. ; LX ; CAL ;Exit paran ; HL WCS: PU: MO OR, JZ	te console string minates when a C ereturn is outpu ed. equence I H,BUFFER _L WCS neters _> 00H byte term SH H V A,M A A WCSX I LF	NOH byte is encountered. ut when a line feed is ninator ;Save buffer pointer ;Get next character ;Check if OOH ;Yes, exit
01F4 4F 01F5 C9 01F5 E5 01F7 7E 01F8 B7 01F9 CA1602 01FC FE0A 01FC CC0C02	RE ;WCS - Wri ;Output ter ;A carriag; ;encounter ;Calling s. ; LX ; CAL ;Exit parau ; HL WCS: PU: MCC 0R, JZ CP	te console string minates when a C ereturn is outpu equence I H,BUFFER L WCS meters -> OOH byte term SH H V A,M A A WCSX I LF WCSLF	NOH byte is encountered. it when a line feed is innator ;Save buffer pointer ;Get next character ;Check if 00H ;Yes, exit ;Check if line feed
01F4 4F 01F5 C9 01F5 C9 01F6 E3 01F7 7E 01F8 B7 01F7 CE0A 01FC FE0A 01FC CC0C02 0201 5F	RE ;WCS - Wri ;Output te ;A carriage ;encounter ;Calling s. ; LX ; CAL ;Exit paran ; HL WCS: PUC MOV OR. JZ CP CZ MOV	te console string minates when a C ereturn is outpu ed. equence I H,BUFFER _L WCS meters -> OOH byte term SH H V A,M A A WCSX I LF WCSLF V E,A	NOH byte is encountered. it when a line feed is finator ;Save buffer pointer ;Get next character ;Check if 00H ;Yes, exit ;Check if line feed ;Yes, output a carriage return ;Character to be output
01F4 4F 01F5 C9 01F5 E5 01F7 7E 01F9 E7 01F9 E7 01F9 CA1602 01FC FE0A 01FC CC0C02 0201 5F 0202 0E06	RE ;WCS - Wri ;Output ter ;A carriag ;encounter ;Calling s ; LX ; CAl ;Exit para ; HL WCS: PUS MO OR JZ CP CZ MO WV	te console string minates when a C e return is outpu ed. equence I H,BUFFER L WCS meters -> 00H byte term SH H V A,M A A WCSLF I LF WCSLF V E,A I C,S\$DIRCONJ	<pre>DOH byte is encountered. it when a line feed is innator ;Save buffer pointer ;Get next character ;Check if 00H ;Yes, exit ;Check if line feed ;Yes, output a carriage return ;Character to be output 0; Function code</pre>
01F4 4F 01F5 C9 01F5 C9 01F6 E3 01F7 7E 01F9 CA1602 01F8 B7 01F9 CA1602 01F2 CC0C02 0201 5F 0202 0E06 0204 CD0500	RE ;WCS - Wri ;Output ten ;A carriag; ;encounterd ;Calling sc ; LX ; LX ; ; CAL ;Exit paran ; HL WCS: WCS: PU: MON OR JZ CP CZ CZ MON CAL CAL CAL CAL CAL CAL CAL CAL CAL CAL	te console string minates when a C ereturn is outpu ed. Equence I H,BUFFER L WCS neters -> OOH byte tern SH H V A,M A A WCSX I LF WCSLF V E,A I C,B&DIRCONJ L BDOS	<pre>DOH byte is encountered. it when a line feed is innator ;Save buffer pointer ;Get next character ;Check if 00H ;Yes, exit ;Check if line feed ;Yes, output a carriage return ;Character to be output 10 ;Function code ;Jurut character</pre>
01F4 4F 01F5 C9 01F5 C9 01F5 75 01F7 75 01F9 CA1602 01FC FE0A 01FC CC0C02 0201 5F 0202 0E06 0204 CD0500 0207 E1	RE ;WCS - Wri ;Output ter ;A carriage ;encounter ;Calling s. ; LX ; CAL ;Exit paran ; HL WCS: PUE MO OR JZ CP CZ MO MV CAL POL POL	te console string minates when a C ereturn is outpu ed. equence I H,BUFFER L WCS neters -> OOH byte term SH H V A,M A A WCSX I LF WCSLF V E,A I C,B#DIRCONI -> H	<pre>DOH byte is encountered. it when a line feed is ininator ;Save buffer pointer ;Get next character ;Check if 00H ;Yes, exit ;Check if line feed ;Yes, output a carriage return ;Character to be output [0 ;Function code ;Output character ;Recover buffer pointer</pre>
01F4 4F 01F5 C9 01F5 C9 01F6 E5 01F7 7E 01F9 CA1602 01F8 B7 01F9 CA1602 01FC FE0A 01F9 CA1602 01FC COC02 0201 5F 0202 0E06 0204 CD0500 0207 E1 0208 23	RE ;WCS - Wri ;Output ten ;A carriag ;encounterd ;Calling sc ; LX ; LX ; LX ; CAL ;Exit param ; HL WCS: WCS: WCS CZ MOV CZ MOV CAL POI IN	te console string minates when a C ereturn is outpu ed. equence I H,BUFFER L WCS meters -> 00H byte term SH H V A,M A A WCSLF V E,A I LF WCSLF V E,A I C,BEDIRCONJ L BDOS P H K H	<pre>DOH byte is encountered. it when a line feed is innator ;Save buffer pointer ;Get next character ;Check if 00H ;Yes, exit ;Check if line feed ;Yes, output a carriage return ;Character to be output [0] ;Function code ;Output character ;Recover buffer pointer ;Update to next char.</pre>
01F4 4F	RE ;WCS - Wri ;Output ter ;A carriage ;encounter ;Calling s. ; LX ; CAL ;Exit paran ; HL WCS: PUE MO OR JZ CP CZ MO MV CAL POL POL	te console string minates when a C ereturn is outpu ed. equence I H,BUFFER L WCS meters -> 00H byte term SH H V A,M A A WCSLF V E,A I LF WCSLF V E,A I C,BEDIRCONJ L BDOS P H K H	<pre>DOH byte is encountered. it when a line feed is ininator ;Save buffer pointer ;Get next character ;Check if 00H ;Yes, exit ;Check if line feed ;Yes, output a carriage return ;Character to be output [0 ;Function code ;Output character ;Recover buffer pointer</pre>
01F4 4F 01F5 C9 01F5 C9 01F6 E5 01F7 7E 01F9 CA1602 01F6 E7 01F9 CA1602 01F6 COC02 021F9 CA1602 01F6 COC02 0201 5F 0202 0E06 0204 CD0500 0207 E1 0208 23	RE ;WCS - Wri ;Output ten ;A carriag ;encounterd ;Calling sc ; LX ; LX ; LX ; CAL ;Exit param ; HL WCS: WCS: WCS CZ MOV CZ MOV CAL POI IN	te console string minates when a C ereturn is outpu ed. equence I H,BUFFER L WCS neters -> OOH byte tern SH H V A,M A A WCSX I LF WCSLF V E,A I C,B&DIRCONJ L BDOS - H K H - WCS	<pre>NOH byte is encountered. It when a line feed is ininator ;Save buffer pointer ;Get next character ;Check if 00H ;Yes, exit ;Check if 00H ;Yes, output a carriage return ;Character to be output IO ;Function code ;Output character ;Recover buffer pointer ;Update to next char. ;Output next char. ;Line feed encountered</pre>

Figure 5-8. (Continued)

20E 1EOD 210 CD0500	MVI CALL	E,CR BDOS	;Output a CR
213 3E0A 215 C9	MVI RET	A,LF	;Recreate line feed ;Output LF
	WCSX:		;Exit
216 E1	POP	н	;Balance the stack
217 C9	RET		

Figure 5-8. (Continued)

Example

Notes

0007 =	B\$GETIO	EQU 7	;Get IOBYTE
0005 =	BDOS	EQU 5	;BDOS entry point
0000 0E07	MVI	C,B\$GETIO	;Function code
0002 CD0500	CALL	BDOS	;A = IOBYTE

Purpose This function places the current value of the IOBYTE in register A.

As we saw in Chapter 4, the IOBYTE is a means of associating CP/M's logical devices (console, reader, punch, and list) with the physical devices supported by a particular BIOS. Use of the IOBYTE is completely optional. CP/M, to quote from the Digital Research *CP/M 2.0 Alteration Guide*, "...tolerate[s] the existence of the IOBYTE at location 0003H."

In practice, the STAT utility provided by Digital Research does have some features that set the IOBYTE to different values from the system console.

Figure 5-9 summarizes the IOBYTE structure. A more detailed description was given in Chapter 4.

Each two-bit field can take on one of four values: 00, 01, 10, and 11. The value can be interpreted by the BIOS to mean a specific physical device, as shown in Table 4-1.

Figure 5-10 has equates that are used to refer to the IOBYTE. You can see that the values shown are declared using the SHL (shift left) operator in the Digital Research Assembler. This is just a reminder that the values are structured this way in the IOBYTE itself.

```
+-----+
Bit No. ! 7 : 6 ! 5 : 4 ! 3 : 2 ! 1 : 0 !
+-----+
Logical Device List Punch Reader Console
```

Figure 5-9. The IOBYTE structure

```
;IOBYTE equates
                 These are for accessing the IOBYTE.
                 ;Mask values to isolate specific devices.
                 ;(These can also be inverted to preserve all BUT the
                 ; specific device)
0003 =
                 IO$CONM EQU
                                   0000$0011B
                                                     :Console mask
                                   0000$1100B
000C ≈
                 IO$RDRM EQU
                                                     ;Reader mask
0030 =
                 IO$PUNM EQU
                                   0011$0000B
                                                     ;Punch mask
00C0 =
                 IO$LSTM EQU
                                   1100$0000B
                                                     ;List mask
                                                     ;Console values
                                                     ;Console -> TTY:
;Console -> CRT:
0000 =
                IO$CTTY EQU
                                   0
                 TO$CCRT EQU
0001 ≈
                                   1
                 IOSCBAT EQU
                                                     ;Console input <- RDR:
;Console output -> LST:
;Console -> UC1: (user console 1)
0002 ≈
                                   2
0003 =
                 IO$CUC1 EQU
                                   з
                                                     ;Reader values
                                   0 SHL 2
1 SHL 2
2 SHL 2
0000 =
                 IOSRITY FOU
                                                     ;Reader <- TTY:
;Reader <- RDR:
0004 =
                 TOSERDE FOLL
                                                      ;Reader <- UR1: (user reader 1)
                 IO$RUR1 EQU
0008 =
000C =
                 IO$RUR2 EQU
                                   3 SHL 2
                                                     ;Reader (- UR2: (user reader 2)
                                                     ;Punch values
0000 =
                 IO$PTTY EQU
                                   O SHL 4
                                                     ; Punch -> TTY:
                                                      ;Punch -> PUN:
0010 =
                 IO$PPUN EQU
                                   1 SHL 4
0020 =
                 IO$PUP1 EQU
                                   2 SHL 4
                                                     ;Punch -> UP1: (user punch 1)
0030 =
                IO$PUP2 EQU
                                   3 SHL 4
                                                     ;Punch -> UP2: (user punch 2)
                                                     ;List values
                                   O SHL 6
                IO$LTTY EQU
                                                     ;List -> TTY:
;List -> CRT:
;List -> LPT: (physical line printer)
0000 \doteq
                                   1 SHL 6
0040 =
                 IO$LCRT EQU
                 IO$LLPT EQU
                                   2 SHL 6
0080 =
0000 =
                 IO$LUL1 EQU
                                   3 SHL 6
                                                     ;List -> UL1: (user list 1)
```

Figure 5-10. IOBYTE equates

Function 8: Set IOBYTE

Function Code: C = 08HEntry Parameters: E = New IOBYTE value Exit Parameters: None

Example This listing shows you how to assign the logical reader device to the BIOS's console driver. It makes use of some equates from Figure 5-10.

0007 = 0008 = 0005 =	B\$GETIO B\$SETIO BDOS	EQU	8 ;Set	IOBYTE IOBYTE entry point
000C = 0008 =	IO\$RDRM IO\$RUR1		0000\$1100B 2 SHL 2	;Reader bit mask ;User reader select
	;This example ;reader to the			
0100 0100 0E07	ORG MVI	100H C,B\$GETI	0 ;Firs	t, get current IOBYTE

0102 CD0500	CALL	BDOS	AND OFFH ;Preserve all but
0105 E6F3	ANI	(NOT IO\$RDRM)	; reader bits
0107 F608 0109 5F 010A 0E08 010C CD0500	ORI MOV MVI CALL	IO\$RUR1 E,A C,B\$SETIO BDOS	;OR in new setting ;Ready for set IOBYTE ;Set new value

Purpose This function sets the IOBYTE to a new value which is given in register E. Because of the individual bit fields in the IOBYTE, you will normally use the Get IOBYTE function, change some bits in the current value, and then call the Set IOBYTE function.

Notes You can use the Set IOBYTE, Get IOBYTE, and Direct Console I/O functions together to create a small program that transforms your computer system into a "smart" terminal. Any data that you type on your keyboard can be sent out of a serial communications line to another computer, and any data received on the line can be sent to the screen.

Figure 5-11 shows this program and illustrates the use of all of these functions. For this program to function correctly, your BIOS must check the IOBYTE and detect whether the logical console is connected to the physical console (with the IOBYTE set to TTY:) or to the input side of the serial communications line (with the IOBYTE set to RDR:).

Figure 5-11 shows how to use the Get and Set IOBYTE functions to make a simple terminal emulator. For this example to work, the BIOS must detect the Console Value as 3 (IO\$CUC1) and connect Console Status, Input, and Output functions to the communications line.

	0006 =	B\$DIRCONIO	EQU 6	;Direct console input/output
	0007 =	B\$GETI0	EQU 7	;Get IOBYTE
	0008 =	B\$SETI0	EQU 8	;Set IOBYTE
	000B =	B\$CONST	EQU 11	;Get console status (sneak preview)
	0005 =	BDOS	EQU 5	;BDOS entry point
	0003 =	IO\$CONM EQU	0000\$0011B	;Console mask for IOBYTE
	0001 =	IO\$CCRT EQU	1	;Console -> CRT:
	0003 =	IO\$CUC1 EQU	3	;Console -> user console #1
		TERM:		
	0000 CD2A00	CALL	SETCRT	;Connect console -> CRT:
		TERM\$CKS:		
	0003 CD5200	CALL	CONST	:Get CRT status
	0006 CA2400	JZ	TERM\$NOK I	No console input
	0009 CD4B00	CALL	CONIN	;Get keyboard character
	000C CD3000	CALL	SETCOMM	:Connect console -> comm. line
	000F CD4500	CALL	CONOUT	;Output to comm. line
		TERM\$CCS:		;Check comm. status
1	0012 CD5200	CALL	CONST	;Get "console" status
1	0015 CA0000	JZ	TERM	;No incoming comm. character

Figure 5-11. Simple terminal emulator

Г				
	001B CD2A00	CALL	SETCRT	;Connect console -> CRT:
l	001E CD4500	CALL JMP	CONOUT TERM\$CKS	;Output to CRT :Loop back to check keyboard status
l	0021 C30300	UMP	I EKN#UND	FLOOP Dack to check keyboard status
l		TERM\$NOKI:		
l	0024 CB3000	CALL	SETCOMM	:Connect console -> comm. line
l	0027 C31200	JMP	TERM\$CCS	;Loop back to check comm. status
l				
l		SETCRT:		;Connect console -> CRT:
l	002A F5	PUSH	PSW	;Save possible data character
l	002B 0601	MVI	B, IO\$CCRT	;Connect console -> CRT:
l	002D C33300	JMP	SETCON	;Common code
l		SETCOMM:		:Connect console -> comm. line
۱	0030 F5	PUSH	PSW	:Save possible data character
l	0031 0603	MVI	B.IO\$CUC1	;Connect console -> comm. line
l	0001 0000		2,101000	Drop into SETCON
I				
I		SETCON:		;Set console device
l			_	New code in B (in bits 1,0)
l	0033 C5	PUSH	B	;Save code
l	0034 OE07	MVI	C,B\$GETIO	;Get current IOBYTE
l	0036 CD0500	CALL	BDOS	
l	0039 E6FC	ANI POP	B	AND OFFH ;Preserve all but console ;Recover required code
ſ	003B C1		В	;Recover required code :OR in new bits
L	003C B0 003D 5F	ORA MOV	E,A	Ready for setting
l	003E 0E08	MVI	C,B\$SETIO	Function code
ł	0032 0200 0040 CD0500	CALL	BDOS	franceron cobe
l	0043 F1	POP	PSW	Recover possible data character:
I	0044 69	RET		,
I				
l		CONOUT:		
I	0045 5F	MOV	E,A	;Get data byte for output
I	0046 OE06	M∨I	C,B\$DIRCONIO	;Function code
I	0048 C30500	JMP	BDOS	;BDOS returns to CONOUT's caller
I		000170		
l	DOAR OFO	CONIN: MVI		Eurotion and
I	004B 0E06 004D 1EFF	MVI	C,B\$DIRCONIO E,OFFH	;Function code ;Indicate console input
I	004D TEFF 004F C30500	JMP	BDOS	BDOS returns to CONIN's caller
I	004, 000000	0.1		, 2202 . Claims to control s carton
I		CONST:		
Į	0052 OEOB	MVI	C,B\$CONST	;Function code
I	0054 CD0500	CALL	BDOS	
I	0057 B7	ORA	A	;Set Z-flag to result
ļ	0058 C9	RET		
I				

Figure 5-11. (Continued)

Function 9: Display "\$"-Terminated String

Function Code:C = 09HEntry Parameters:DE = Address of first byte of stringExit Parameters:None

Example

0009 =	B\$PRINTS	EQU	9	;Print \$-Terminated String
0005 =	BDOS	EQU	5	;BDOS entry point
000D =	CR	EQU	0DH	;Carriage return
000A =	LF	EQU	0AH	;Line feed
0009 =	TAB	EQU	09H	;Horizontal tab

0000	0D0A095468MESSAGE	:	DB	CR,LF,T	AB, This is	a message',CR,LF,'\$'
0019	0E09 110000 CD0500	MVI LXI CALL	C,B\$PRI D,MESSA BDOS		;Function ;Pointer t	

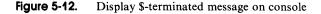
Purpose This function outputs a string of characters to the console device. The address of this string is in registers DE. You must make sure that the last character of the string is "\$"; the BDOS uses this character as a marker for the end of the string. The "\$" itself does not get output to the console.

While the BDOS is outputting the string, it expands tabs as previously described, checks to see if there is an incoming character, and checks for CONTROL-S (XOFF, which stops the output until another character is entered) or CONTROL-P (which turns on or off echoing of console characters to the printer).

Notes One of the biggest drawbacks of this function is its use of "\$" as a terminating character. As a result, you cannot output a string with a "\$" in it. To be truly general-purpose, it would be better to use a subroutine that used an ASCII NUL (00H) character as a terminator, and simply make repetitive calls to the BDOS CONOUT function (code 2). Figure 5-3 is an example of such a subroutine.

Figure 5-12 shows an example of a subroutine that outputs one of several messages. It selects the message based on a message code that you give it as a parameter. Therefore, it is useful for handling error messages; the calling code can pass it an 8-bit error code. You may find it more flexible to convert this subroutine to using 00H-byte-terminated messages using the techniques shown in Figure 5-3.

```
;OM (Output message)
This subroutine selects one of several messages based on
; the contents of the A register on entry. It then displays
; this message on the console.
;Each message is declared with a "$" as its last character.
; If the A register contains a value larger than the number
; of messages declared, OM will output "Unknown Message".
;As an option, OM can output carriage return / line feed
; prior to outputting the message text.
;Entry parameters
        HL -> message table
                   This has the form :
                      DB
                                           ;Number of messages in table
                                 з
                                 MSGO
                                           ;Address of text (A = 0)
                      DW
                                            ;(A = 1)
                      DW
                                 MSG1
                      nω
                                 MSG2
                                            ; (A = 2)
          MSGO:
                     DB
                                 'Message text$
                                 ...etc.
                     A = Message code (from 0 on up)
B = Output CR/LF if non-zero
```



Calling sequence ; H, MSG\$TABLE LXI ; LDA MSGCODE ; Suppress CR/LF MVI B. 0 CALL OM 9 ;Print \$-terminated string 0009 = **B**\$PRINTS EQU 5 ;BDOS entry point 0005 = BDOS EQU 000D = CR EQU ODH ;Carriage return 000A = LF EQU OAH ;Line feed CR, LF, 1\$1 0000 0D0A24 OM\$CRLF: DB. 'Unknown Message\$' 0003 556E6B6E6F0M\$UM: DB OM: PSW ;Save message code PUSH 0013 F5 PUSH ;Save message table pointer 0014 E5 н 0015 78 MOV A,B ;Check if CR/LF required 0016 B7 ORA Α ;No 0017 CA2200 JZ OM\$NOCR 001A 110000 LXI D, OM\$CRLF ;Output CR/LF 001D 0E09 001F CD0500 MVI C, B\$PRINTS CALL BDOS. OM\$NOCR: ;Recover message table pointer POP 0022 E1 н POP PSW ;Recover message code 0023 F1 0024 BE ;Compare message to max. value CMP м 0025 D23700 JNC OM\$ERR ;Error-code not <= max. 0028 23 INX н ;Bypass max. value in table 0029 87 ADD Α ;Message code * 2 ;Make (code * 2) a word value 002A 5F MOV E,A 002B 1600 MVI D, 0 002D 19 DAD D ;HL -> address of message text Ē,M ;Get LS byte 002E 5E MOV 002F 23 INX ;HL -> MS byte 0030 56 MOV D, M ;Get MS byte ;DE -> message text itself OM\$PS: ;Print string entry point C.B\$PRINTS 0031 0E09 MVT ;Function code 0033 CD0500 CALL BDOS 0036 C9 RET ;Return to caller OM\$ERR: ;Error 0037 110300 LXI D,OM\$UM ;Point to "Unknown Message" 003A C33100 JMP OM\$PS ;Print string

Figure 5-12. (Continued)

Function 10: Read Console String

Function Code:	C = 0AH
Entry Parameters:	DE = Address of string buffer
Exit Parameters:	String buffer with console bytes in it

Example

000A =	B\$READCONS	EQU	10	;Read Console String
0005 =	BDOS	EQU	5	;BDOS entry point

Chapter 5: The Basic Disk Operating System **91**

0050 =	BUFLEN	EQU	80	;Buffer length
0000 50	BUFFER: BUFMAXCH:	DB	BUFLEN	;Console input buffer ;Max. no. of characters in ; buffer
0001 00 0002	BUFACTCH: BUFCH:	DB DS	0 BUFLEN	;Actual no. of characters input ;Buffer characters
0052 0E0A 0054 110000 0057 CD0500	MVI LXI CALL	C,B\$REA D,BUFFE BDOS		;Function code ;Pointer to buffer

Purpose

This function reads a string of characters from the console device and stores them in a buffer (address in DE) that you define. Full line editing is possible: the operator can backspace, cancel the line and start over, and use all the normal control functions. What you will ultimately see in the buffer is the final version of the character string entered, without any of the errors or control characters used to do the line editing.

The buffer that you define has a special format. The first byte in the buffer tells the BDOS the maximum number of characters to be accepted. The second byte is reserved for the BDOS to tell you how many characters were actually placed in the buffer. The following bytes contain the characters of the string.

Character input will cease either when a CARRIAGE RETURN is entered or when the maximum number of characters, as specified in the buffer, has been received. The CARRIAGE RETURN is not stored in the buffer as a character—it just serves as a terminator.

If the first character entered is a CARRIAGE RETURN, then the BDOS sets the "characters input" byte to 0. If you attempt to input more than the maximum number of characters, the "characters input" count will be the same as the maximum value allowed.

Notes

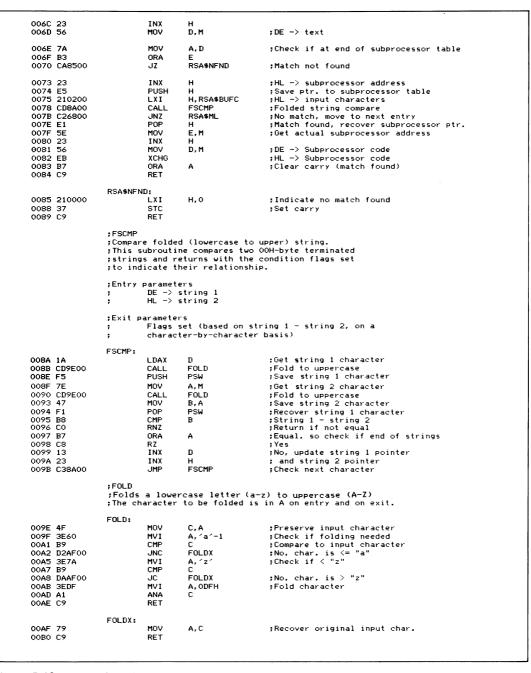
This function is useful for accepting console input, especially because of the line editing that it allows. It should be used even for single-character responses, such as "Y/N" (yes or no), because the operator can type "Y", backspace, and overtype with "N". This makes for more "forgiving" programs, tolerant of humans who change their minds.

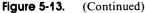
Figure 5-13 shows an example subroutine that uses this function. It accepts console input, matches the input against a table, and transfers control to the appropriate subroutine. Many interactive programs need to do this; they accept an operator command and then transfer control to the appropriate command processor to deal with that command.

This example also includes two other subroutines that are useful in their own right. One compares null-byte-terminated strings (FSCMP), and the other converts, or "folds," lowercase letters to uppercase (FOLD).

```
: RSA
                  ;Return subprocessor address
                  This subroutine returns one of several addresses selected; from a table by matching keyboard input against specified;
                  ; strings. It is normally used to switch control to a
; particular subprocessor according to an option entered
; by the operator from the keyboard.
                  ;Character string comparisons are performed with case-folding;
                  ; that is, lowercase letters are converted to uppercase.
                  ; If the operator input fails to match any of the specified
                  ; strings, then the carry flag is set. Otherwise, it is
                  ; cleared.
                  ;Entry parameters
                           HL -> Subprocessor select table
                  .
                                     This has the form :
DW TEXTO, SUBPROCO
                                     DW
                                               TEXT1, SUBPROC1
                                     nu
                                               Ω
                                               0 ;Terminator
'add',0 ;OOH-byte terminated
                  ;
                           TEXTO:
                                     DB
                  ;
                                               'subtract',0
                                     DB
                  ;
                           TEXT1:
                           SUBPROCO:
                  :
                                     Code for processing ADD function.
                  :
                           SUBPROC1:
                  ;
                                     Code for processing SUBTRACT function.
                  :
                  :Exit parameters
                           DE -> operator input string (OOH-terminated
                  ;
                                  input string).
                  .
                           Carry Clear, HL -> subprocessor.
                  ;
                           Carry Set, HL = 0000H.
                  :
                  Calling sequence
                                     H, SUBPROCTAB
                                                        Subprocessor table
                           CALL
                                     RSA
                  .
                           JC
                                     ERROR
                                                        ;Carry set only on error
;Fake CALL instruction
                  :
                           LXI
                                     D. RETURN
                  :
                           PUSH
                                     D
                                                        ;Push return address on stack
                  :
                                                        ;"CALL" to subprocessor
                           PCHL
                  :
                           RETURN:
                  :
000A =
                  B$READCONS
                                     EQU
                                               10
                                                        ;Read console string into buffer
0005 =
                  BDOS
                                     EQU
                                               5
                                                        ;BDOS entry point
0050 =
                  RSA$BL
                                     EQU
                                               80
                                                         ;Buffer length
0000 50
                  RSA$BUF:
                                     DB
                                               RSA$BL
                                                        ;Max. no. of characters
0001 00
                  RSA$ACTC:
                                     DB.
                                               0
                                                         ;Actual no. of characters
                                               RSA$BL
                                                        ;Buffer characters
;Safety terminator
0002
                  RSA$BUEC:
                                     DS
0052 00
                                     DB
                                               0
                  RSA:
0053 2B
                           DCX
                                     н
                                                         ;Adjust Subprocessor pointer
0054 2B
                           DCX
                                     н
                                                         ; for code below
0055 E5
                           PUSH
                                     н
                                                         ;Top of stack (TOS) -> subproc. table - 2
0056 0E0A
                           MVI
                                     C, B$READCONS
                                                         ;Function code
0058 110000
                            LXI
                                     D.RSA$BUF
                                                         :DE -> buffer
005B CD0500
                                     BDOS
                            CALL
                                                         ;Read operator input and
                                                         ; Convert to OOH-terminated
005E 210100
                           LXI
                                     H,RSA$ACTC
                                                         ;HL -> actual no. of chars. input
0061 5E
                           MOV
                                     E,M
                                                         ;Get actual no. of chars. input
0062 1600
                           MVI
                                     D,0
                                                         ;Make into word value
                                                         ;HL -> first data character
;HL -> first UNUSED character in buffer
0064 23
                            TNX
                                     н
                           DAD
                                     D
                                                         ;Make input buffer OOH terminated
0066 3600
                                     M. 0
                           MVI
                  RSA$ML:
                                                         :Compare input to specified values
                                                         : Main loop
                                                         ;Recover subprocessor table pointer
;Move to top of next entry
;HL -> text address
0068 E1
                           POP
                                     н
0069 23
                            INX
                                     н
006A 23
                            INX
                                     н
006B 5E
                                     E,M
                                                         ;Get text address
                            MOV
```

Figure 5-13. Read console string for keyboard options





Function 11: Read Console Status

Function Code: C = 0BHEntry Parameters: None Exit Parameters: A = 00H if no incoming data byte A = 0FFH if incoming data byte

Example

000B =	B\$CONST	EQU 11	;Get Console Status
0005 =	BDOS	EQU 5	;BDOS entry point
0000 0E0B 0002 CD0500	MVI CALL	C,B\$CONST BDOS	;Function code ;A = 00 if no character waiting ;A = 0FFH if character waiting

- **Purpose** This function tells you whether a console input character is waiting to be processed. Unlike the Console Input functions, which will wait until there is input, this function simply checks and returns immediately.
- **Notes** Use this function wherever you want to interrupt an executing program if a console keyboard character is entered. Just put a Console Status call in the main loop of the program. Then, if the program detects that keyboard data is waiting, it can take the appropriate action. Normally this would be to jump to location 0000H, thereby aborting the current program and initiating a warm boot. Figure 5-11 is an example subroutine that shows how to use this function.

Function 12: Get CP/M Number

Function Code: C = 0CHEntry Parameters: None Exit Parameters: HL = Version number code

Example

000C =	B\$GETVER	EQU 12	;Get CP/M Version Number
0005 =	BDOS	EQU 5	;BDOS entry point
0000 0E0C 0002 CD0500	MVI CALL	C,B\$GETVER BDOS	;Function code ;H = 00 for CP/M ;L = version (e.g. 22H for 2.2)

Purpose This function tells you which version of CP/M you are currently running. A two-byte value is returned:

H = 00H for CP/M, H = 01H for MP/M

- L = 00H for all releases before CP/M 2.0
- L = 20H for CP/M 2.0, 21 H for 2.1, 22 H for 2.2, and so on for any subsequent releases.

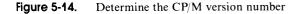
This information is of interest only if your program has some version-specific logic built into it. For example, CP/M version 1.4 does not support the same Random File Input/Output operations that CP/M 2.2 does. Therefore, if your program uses Random I/O, put this check at the beginning to ensure that it is indeed running under the appropriate version of CP/M.

Notes Figure 5-14 is a subroutine that checks the current CP/M version number, and, if it is not CP/M 2.2, displays an explanatory message on the console and does a warm boot by jumping to location 0000H.

Function 13: Reset Disk System

Function Code: C = 0DH Entry Parameters: None Exit Parameters: None

		;This s ;operat	ing syst	em and,	if not	e version number of the CP/M version 2, displays a warm boot.
		;Entry ;	and exit None	paramet	ers	
		:Callir	ng sequen	ce		
		;	CALL	CCPM	;Warm	boots if not CP/M 2
0009	=	B\$PRINT	s	EQU	9	;Display \$-terminated string
0000	=	B\$GETVE		EQU	12	Get version number
0005		BDOS		EQU	5	;BDOS entry point
0000		CR		EQU	ODH	;Carriage return
000A	=	LF		EQU	OAH	;Line feed
0000	ODOA	CCPMM:	DB	CR,LF		
	5468697320)	DB			can only run under CP/M version 2.4
0031	0D0A24		DB	CR,LF,′	\$ ⁻	
		CCPM:				
0034	OEOC		MVI	C,B\$GET	VER	;Get version number
0036	CD0500		CALL	BDOS		
0039	7C		MOV	А,Н		;H must be O for CP/M
003A	B7		ORA	A		
003B	C24700		JNZ	COPME		:Must be MP/M
003E	7D		MOV	A,L		;L = version number of CP/M
003F	E6F0		ANI	OFOH		Version number in MS nibble
0041	FE20		CPI	20H		:Check if version 2
0043	C24700		JNZ	CCPME		Must be an earlier version
0046	C:9		RET			;Yes, CP/M version 2
		CCPME:				;Error
0047	0E09		MVI	C,B\$PRI	NTS	;Display error message
0049	110000		LXI	D. CCPMM		· · · · · · · · · · · · · · · · · · ·
004C	CD0500		CALL	BDOS		
	C30000		JMP	0		:Warm boot



96 The CP/M Programmer's Handbook

Example

000D =	B\$DSKRESET	EQU	13	;Reset Disk System
0005 =	BDOS	EQU	5	;BDOS entry point
0000 0E0D 0002 CD0500	MVI CALL	C,B\$DSKI BDOS	RESET	;Function code

Purpose This function requests CP/M to completely reset the disk file system. CP/M then resets its internal tables, selects logical disk A as the default disk, resets the DMA address back to 0080H (the address of the buffer used by the BDOS to read and write to the disk), and marks all logical disks as having Read/Write status.

The BDOS will then have to log in each logical disk as each disk is accessed. This involves reading the entire file directory for the disk and rebuilding the allocation vectors (which keep track of which allocation blocks are free and which are used for file storage).

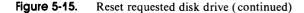
Notes This function lets you change the diskettes under program control. If the operator were to simply change diskettes, without CP/M knowing about it, the next access to the (now different) diskette would force CP/M to declare the disk Read-Only, thwarting any further attempts to write on the diskette. If you need to reset one or two disks, rather than the entire disk system, look ahead to the Reset Disk function (code 37) described at the end of this chapter.

Figure 5-15 shows a simple subroutine that outputs a message on the console, requesting that the diskette in a specified drive be changed. It then issues a Reset Disk function call to make sure that CP/M will log in the diskette on the next access to the drive.

```
;CDISK
                ;Change disk
                ;This subroutine displays a message requesting the
                 ;user to change the specified logical disk, then waits
                ; for a carriage return to be pressed. It then issues
; a Disk Reset and returns to the caller.
                Entry parameters
                          A = Logical disk to be changed (A = 0, B = 1)
                :
                ;Exit parameters
                          None
                :
                 :Calling sequence
                          MUT
                                   A. 0
                                                     :Change drive A:
                          CALL
                                   CDISK
                 :
000D =
                 B$DSKRESET
                                   EQU
                                            13
                                                     ;Disk Reset function code
0009 =
                                            9
                                                     ;Print $-terminated string
                 B$PRINTS
                                   EQU
                                                     ;Get console input
0001 =
                 B$CONIN
                                   EQU
                                            1
0005 =
                 BDOS
                                   EQU
                                            5
                                                     ;BDOS entry point
```

Figure 5-15. Reset requested disk drive

```
000D =
                 CR
                                  EQU
                                            ODH
                LF
                                  EQU
                                            OAH
000A =
0000 0D0A436861CDISKM:
                                  DB
                                            CR,LF, 'Change logical disk '
0016 00 0
0017 3A20616E64
                CDISKD:
                                  DB
                                   DB
                                            ': and press Carriage Return to continue$'
                CDISK:
003E_C640
                          ADI
                                   'A'-1
                                                     Convert to letter
0041 321600
                                  CDISKD
                          STA
                                                     ;Store in message
                                  C, B$PRINTS
D, CDISKM
0044 0E09
0046 110000
                         MUT
                                                     ;Display message
                          LXI
0049 CD0500
                                  BDOS
                          CALL
                CDISKW:
004C 0E01
                         MVI
                                  C, B$CONIN
                                                     ;Get keyboard character
004E CD0500
                          CALL
                                  BDOS
0051 FEOD
                          CPI
                                   CR
0053 C24C00
                          JNZ
                                  CDISKW
0056 0E0D
                          MVI
                                  C, B$DSKRESET
                                                     ;Now reset disk system
0058 CD0500
                          CALL
                                  BDOS
005B C9
                         RET
```



Function 14: Select Logical Disk

Function Code:	C = 0EH
Entry Parameters:	E = Logical Disk Code
	00H = Drive A
	01 H = Drive B and so on
Exit Parameters:	None

Example

000E =	B\$SELDSK	EQU 14	;Select Logical Disk
0005 =	BDOS	EQU 5	;BDOS entry point
0000 0E0E 0002 1E00 0004 CD0500	MVI MVI CALL	C,B\$SELDSK E,O BDOS	;Function code ;E = 0 for A:, 1 for B: etc.

Purpose This function makes the logical disk named in register E the default disk. All subsequent references to disk files that do not specify the disk will use this default. When you reference a disk file that *does* have an explicit logical disk in its name you do not have to issue another Select Disk function; the BDOS will take care of that for you.

Notes Notice the way in which the logical disk is specified in register E. It is not the same as the disk drive specification in the first byte of the file control block. In the FCB, a value of 00H is used to mean "use the current default disk" (as specified in the last Select Disk call or by the operator on the console). With this function, a

98 The CP/M Programmer's Handbook

value of 00H in register A means that A is the selected drive, a value of 01H means drive B, and so on to 0FH for drive P, allowing 16 drives in the system.

If you select a logical disk that does not exist in your computer system, the BDOS will display the following message:

BDOS Err on J: Select

If you type a CARRIAGE RETURN in order to proceed, the BDOS will do a warm boot and transfer control back to the CCP. To avoid this, you must rely on the computer operator not to specify nonexistent disks or build into your program the knowledge of how many logical disk drives are on the system.

Another problem with this function is that you cannot distinguish a logical disk for which the appropriate tables have been built into the BIOS, but for which there is no physical disk drive. The BDOS does not check to see if the drive is physically present when you make the Select Disk call. It merely sets up some internal values ready to access the logical disk. If you then attempt to access this nonexistent drive, the BIOS will detect the error. What happens next is completely up to the BIOS. The standard BIOS will return control to the BDOS, indicating an error condition. The BDOS will output the message

BDOS Err on C: Bad Sector

You then have a choice. You can press CARRIAGE RETURN, in which case the BDOS will ignore the error and attempt to continue with whatever appears to have been read in. Or you can enter a CONTROL-C, causing the program to abort and CP/M to perform a warm boot.

Note that the Select Disk function does not return any values. If your program gets control back, you can assume that the logical disk you asked for at least has tables declared for it.

Function 15: Open File

Function Code:C = 0FHEntry Parameters:DE = Address of file control blockExit Parameters:A = Directory code

Example

000F	=	B\$OPEN	EQU	15	;Open File
0005	=	BDOS	EQU	5	;BDOS entry point
		FCB:			;File control block
0000	00	FCB\$DISK:	DB	0	;Search on default disk drive
0001	4649404546	EFCB\$NAME:	DB	'FILENA	ME' ;File name
0009	545950	FCB\$TYP:	DB	TYP'	;File type
000C	00	FCB\$EXTENT:	DB	0	Extent
000D	0000	FCB\$RESV:	DB	0,0	Reserved for CP/M
000F	00	FCB\$RECUSED:	DB	o	Records used in this extent
0010	0000000000000	OFCB\$ABUSED:	DB	0,0,0,0	,0,0,0,0 ;Allocation blocks used
0018	0000000000	0	DB	0,0,0,0	,0,0,0,0
0020	00	FCB\$SEQREC:	DB	0	;Sequential rec. to read/write

0021 0000 0023 00	FCB\$RANREC: FCB\$RANRECO:	DW O DB O	;Random rec. to read/write ;Random rec. overflow byte (MS)
0024 0E0F	MVI	C,B\$OPEN	;Function code
0026 110000	LXI	D,FCB	;DE -> File control block
0029 CD0500	CALL	BDOS	;A = OFFH if file not found

PurposeThis function opens a specified file for reading or writing. The FCB, whose
address must be in register DE, tells CP/M the user number, the logical disk, the
file name, and the file type. All other bytes of the FCB will normally be set to 0.
The code returned by the BDOS in register A indicates whether the file has
been opened successfully. If A contains 0FFH, then the BDOS was unable to find
the correct entry in the directory. If A = 0, 1, 2, or 3, then the file has been opened.

Notes

The Open File function searches the entire file directory on the specified logical disk looking for the file name, type, and extent specified in the FCB; that is, it is looking for an exact match for bytes 1 through 14 of the FCB. The file name and type may be ambiguous; that is, they may contain "?" characters. In this case, the BDOS will open the first file in the directory that matches the ambiguous name in the FCB. If the file name or type is shorter than eight or three characters respectively, then the remaining characters must be filled with blanks.

When the BDOS searches the file directory, it expects to find an *exact* match with each character of the file name and type, including lowercase letters or nongraphic characters. However, the BDOS uses only the least significant seven bits of each character—the most significant bit is used to indicate special file status characteristics, or *attributes*.

By matching the file extent as well as the name and type, you can, if you wish, open the file at some point other than its beginning. For normal sequential access, you would not usually want to do this, but if your program can predict which file extent is required, this is a method of moving directly to it.

It is also possible to open the same file more than once. Each instance requires a separate FCB. The BDOS is not aware that this is happening. It is really only safe to do this when you are reading the file. Each FCB can be used to read the file independently.

Once the file has been found in the directory, the number of records and the allocation blocks used are copied from the directory entry into the FCB (bytes 16 through 31). If the file is to be accessed sequentially from the beginning of the file, the current record (byte 32) must be set to zero by your program.

The value returned in register A is the relative directory entry number of the entry that matched the FCB. As previously explained, the buffer that CP/M uses holds a 128-byte record from the directory with four directory entries numbered 0, 1, 2, and 3. This *directory code* is returned by almost all of the file-related BDOS functions, but under normal circumstances you will be concerned only with whether the value returned in A is 0FFH or not.

Figure 5-16 shows a subroutine that takes a 00H-byte terminated character

string, creates a valid FCB, and then opens the specified file. Shown as part of this example is the subroutine BF (Build FCB). It performs the brunt of the work of converting a string of ASCII characters into an FCB-style disk, file name, and type.

```
; OPENF
                     ;Open File
                     ;Given a pointer to a OOH-byte-terminated file name,
;and an area that can be used for a file control
;block, this subroutine builds a valid file control
;block and attempts to open the file.
                     ;If the file is opened, it returns with the carry flag clear.
;If the file cannot be opened, this subroutine returns
                     ; with the carry flag set.
                     ;Entry parameters
                                DE -> 36-byte area for file control block
                                HL -> OOH-byte terminated file name of the form {disk:} Name {.typ}
                     :
                     •
                                           (disk and typ are optional)
                     :
                     ;Exit parameters
                                Carry clear : File opened correctly.
                                Carry set
                                               : File not opened.
                     :
                     ;Calling Sequence
                                           D,FCB
                                LXI
                                LXI
                                           H. FNAME
                                           OPENF
                                CALL
                                JC
                                           ERROR
                     :
                     ; where
                                                                  ;Space for file control block
                                DS
                     :FCB:
                                           36
                     ;FNAME: DB
                                           'A: TESTFILE. DAT', O
000F =
                     B$OPEN
                                           EQU
                                                                  ;File Open function code
                                                       15
0005 =
                     BDOS
                                           EQU
                                                                  ;BDOS entry point
                                                       5
                     OPENE:
                                                                  ;Preserve pointer to FCB
0000 D5
                                PUSH
                                           D
                                           BF
0001 CD0C00
0004 0E0F
                                CALL
MVI
                                                                  ;Build file control block
                                           C, B$OPEN
0006 D1
0007 CD0500
                                POP
                                                                  Recover pointer to FCB
                                           D
                                CALL
                                           BDOS
000A 17
                                                                  ; If A=OFFH, carry set
                                RAL
                                                                  ;otherwise carry clear
000B C9
                                RET
                     ;BF
                     Build file control block
                     ;This subroutine formats a OOH-byte-terminated string
                     ;(presumed to be a file name) into an FCB, setting
;the disk and file name and type and clearing the
;remainder of the FCB to O's.
                     ;Entry parameters
; DE -> file control block (36 Bytes)
; HL -> file name string (00H-byte-terminated)
                     ;Exit parameters
                                The built file control block
                     ;Calling sequence
; LXI D,FCB
                                           H, FILENAME
                                LXI
                     ;
                                CALL
                                           BF
                     :
                     BF:
```

Figure 5-16. Open file request

0000	23		INX	н	;Check if 2nd char. is ":"
OOOD	7E		MOV	A, M	;Get character from file name
000E			DCX	н	;HL -> now back at 1st char.
000F			CPI	11	;If ":", then disk specified
0011	C21C00		JNZ MOV	BF\$ND A,M	;No disk
0015	E61F		ANI	0001\$1111B	;Get disk letter ;A (41H) -> 1, B (42H) -> 2
0017	23		INX	н	;Bypass disk letter
0018			INX	н	;Bypass ":"
0019	C31D00		JMP	BF\$SD	;Store disk in FCB
		BF\$ND:			;No disk present
001C	AF		XRA	Α	;Indicate default disk
001D	10	BF\$SD:	STAX	D	- Stove disk is FCB
001E			INX	D	;Store disk in FCB ;DE -> 1st char. of name in FCB
001F			MVI	C,8	;File name length
0021	CD3700		CALL	BF\$GT	;Get token
					;Note at this point, BF\$GT
					;will have advanced the string ;pointer to either a "." or
					;00H byte
0024	FE2E		CPI	1.1	;Check terminating character
0026	C22A00		JNZ	BF\$NT	;No file type specified ;Bypass "." in file name
0029	23		INX	н	;Bypass "." in file name
		BF\$NT:			
002A	0E03	24 47411	MVI	С,З	;File type length
002C	CD3700		CALL	BF\$GT	;Get token
					;Note if no file type is
					present BF\$GT will merely
002F	0600		MVI	в,0	;spacefill the FCB ;O-fill the remainder of the FCB
0031			MVI	C, 24	;36 - 12 (disk, name, type = 12 chars.)
0033	CD6400		CALL	BF\$FT	;Re-use fill token S/R
0036	C9		RET		
		;BF\$GT			
			FCB g	et token	
				e scans a file na	
				ters into a file a terminator cl	control block. haracter ("." or OOH),
				of the token is a	
		;If an "*" is encountered, the remainder of the token			
		; is filled with "?".			
		;Entry parameters ; DE -> Into file control block			
		; HL -> Into file name string			
		; C = Maximum no. of characters in token			
	;Exit parameters				
		; File control block contains next token			
		;	A = ler	minating charact	er
		BF\$GT:			
0037			MOV	Α,Μ	;Get next string character
0038			ORA	A	;Check if end of string
	CA5700		JZ CPI	BF\$SFT '*'	;Yes, space fill token ;Check if ?-fill required
003C 003E	CA5COO		JZ	BF\$QFT	;Check if ?-fill required ;Yes, fill with ?
0041			CPI	· · ·	Assume current token is file
					;name
					Check if file type coming up
					;(If current token is file ;type this check is
					;type this check is ;benignly redundant)
0043	CA5700		JZ	BF\$SFT	;Yes, space fill token
0046			STAX	D	;None of the above, so store
			THIN	P	; in FCB
0047			INX INX	D H	;Update FCB pointer ;Update string pointer
0040				••	jopate string pointer

Figure 5-16. (Continued)

0049 OD	DCR	С	;Countdown on token length
004A C23700	JNZ	BF\$GT	Still more characters to go
	BF\$SKIP:		;Skip chars. until "." or OOH
004D 7E	MOV	A. M	Get next string character
004E B7	ORA	A	;Check if OOH
004F C8	RZ		;Yes
0050 FE2E	CPI	1.1	Check if "."
0052 C8	RZ	-	:Yes
0053 23	INX	н	;Update string pointer (only)
0054 C34D00	JMP	BF\$SKIP	;Try next character
	BF\$SFT:		Space fill token
0057 0620	MVI	B, 1 1	
0059 C36400	JMP	BF\$FT	:Common fill token code
			;BF\$FT returns to caller
	BF\$QFT:		Question mark fill token
005C 063F	MVI	B,1?1	
005E CD6400	CALL	BF\$FT	:Common fill token code
0061 C34D00	JMP	BF\$SKIP	Bypass multiple "*" etc.
	BF\$FT:		;Fill token
0064 F5	PUSH	PSW	Save terminating character
0065 78	MOV	Α,Β	;Get fill characer
	BF\$FTL:		;Inner loop
0066 12	STAX	D	Store in FCB
0067 13	INX	D	Update FCB Pointer
0068 OD	DCR	č	Downdate residual count
0069 026600	JNZ	BF\$FTL	;Keep going
006C F1	POP	PSW	Recover terminating character
0060 09	RET		,

Figure 5-16. (Continued)

Function 16: Close File

Function Code:	C = 10H
Entry Parameters:	DE = Address of file control block
Exit Parameters:	A = Directory code

Example

0010 = 0005 =	B\$CLOSE BDOS	EQU EQU	16 5	;Close File ;BDOS entry point
0000	FCB:	DS	36	;File control block
0024 0E10 0026 110000 0029 CB0500	MVI LXI CALL	C,B\$CL D,FCB BDOS	OSE	;Function code ;DE -> File control block ;A = 0,1,2,3 if successful ;A = 0FFH if file name not ; in directory

Purpose This function terminates the processing of a file to which you have written information. Under CP/M you do not need to close a file that you have been reading. However, if you ever intend for your program to function correctly under MP/M (the multi-user version of CP/M) you should close all files regardless of their use.

The Close File function, like Open File, returns a directory code in the A register. Register A will contain 0FFH if the BDOS could not close the file successfully. If A is 0, 1, 2, or 3, then the file has been closed.

Notes When the BDOS closes a file to which data has been written, it writes the current contents of the FCB out to the disk directory, updating an existing directory entry by matching the disk, name, type, and extent number in the same manner that the Open File function does.

Note that the BDOS does not transfer the last record of the file to the disk during the close operation. It merely updates the file directory. You must arrange to flush any partly filled record to the disk. If the file that you have created is a standard CP/M ASCII text file, you must arrange to fill the unused portion of the record with the standard 1AH end-of-file characters as CP/M expects, as explained in the section on the Write Sequential function (code 21).

Function 17: Search for First Name Match

Function Code:C = 11 HEntry Parameters:DE = Address of file control blockExit Parameters:A = Directory code

Example

0011 = B\$SEARCHF EQU 17 ; Search 0005 = BD0S EQU 5 : BD0S e	n First entry point
0003 - BD03 EQ0 3 ;BD03 e	
FCB: ;File o	control block
0000 00 FCB\$DISK: DB 0 ;Search	n on default disk drive
0001 46494C453FFCB\$NAME: DB 'FILE????'	;Ambiguous file name
0009 543F50 FCB\$TYP: DB 'T?P' ;Ambigu	lous file type
000C 00 FCB\$EXTENT: DB 0 ;Extent	1
000D 0000 FCB\$RESV: DB 0,0 ;Reserv	/ed for CP/M
000F 00 FCB\$RECUSED: DB 0 ;Record	is used in this extent
0010 00000000FCB\$ABUSED: DB 0,0,0,0,0,0,0,0);Allocation blocks used
0018 0000000000 DB 0,0,0,0,0,0,0,0)
0020 00 FCB\$SEQREC: DB 0 ;Sequer	ntial rec. to read/write
0021 0000 FCB\$RANREC: DW 0 ;Random	rec. to read/write
0023 00 FCB\$RANRECO: DB 0 ;Random	rec. overflow byte (MS)
0024 #0E11 MVI C.B\$SEARCHF ;Functi	on code
0026 110000 LXI D,FCB ;DE ->	File control block
0029 CD0500 CALL BD0S ;A = 0,	1,2,3.
;(A * 3	32) + DMA -> directory
; entry	
;A = OF	FH if file name not
; found	l

Purpose

This function scans down the file directory for the first entry that matches the file name, type, and extent in the FCB addressed by DE. The file name, type, and extent may contain a "?" (ASCII 3FH) in one or more character positions. Where a "?" occurs, the BDOS will match *any* character in the corresponding position in the file directory. This is known as ambiguous file name matching.

The first byte of an FCB normally contains the logical disk number code. A value of 0 indicates the default disk, while 1 means disk A, 2 is B, and so on up to a

Notes

possible maximum of 16 for disk P. However, if this byte contains a "?", the BDOS will search the default logical disk and will match the file name and type regardless of the user number. This function is normally used in conjunction with the Search Next function (which is described immediately after this function). Search First, in the process of matching a file, leaves certain variables in the BDOS set, ready for a subsequent Search Next.

Both Search First and Search Next return a directory code in the A register. With Search First, A = 0FFH when no files match the FCB; if a file match is found, A will have a value of 0, 1, 2, or 3.

To locate the particular directory entry that either the Search First or Search Next function matched, multiply the directory code returned in A by the length of a directory entry (32 bytes). This is easily done by adding the A register to itself five times (see the code in Figure 5-17 near the label GNFC). Then add the DMA address to get the actual address where the matched directory entry is stored.

There are many occasions when you may need to write a program that will accept an ambiguous file name and operate on all of the file names that match it. (The DIR and ERA commands built into the CCP are examples that use ambiguous file names.) To do this, you must use several BDOS functions: the Set DMA Address function (code 26, described later in this chapter), this function (Search First), and Search Next (code 18). All of this is shown in the subroutine given in Figure 5-17.

```
: GNF
;This subroutine returns an FCB setup with either the
;first file matched by an ambiguous file name, or (if
;specified by entry parameter) the next file name.
;Note : this subroutine is context sensitive. You must
         not have more than one ambiguous file name
sequence in process at any given time.
:
:>>>
         Warning : This subroutine changes the DMA address
.>>>
                      inside the BDOS.
;Entry parameters
         DE -> Possibly ambiguous file name
.
                    (00-byte terminated)
.
                    (Only needed for FIRST request)
;
         HL -> File control block
         A = 0 : Return FIRST file name that matches
= NZ : Return NEXT file name that matches
:
;Exit parameters
;Carry set : A = FF, no file name matches
; A not = OFFH, error in input file name
;Carry clear : FCB setup with next name
             HL -> Directory entry returned
by Search First/Next
;Calling sequence
          LXI
                    D, FILENAME
         LXI
                    H, FCB
```

Figure 5-17. Search first/next calls for ambiguous file name

	; MVI ; CALL	A,O ;or MVI GNF	A,1 for NEXT
0011 = 0012 = 001A = 0005 =	B\$SEARCHF B\$SEARCHN B\$SETDMA BDOS	EQU 17 EQU 18 EQU 26 EQU 5	;Search for first file name ;Search for next file name ;Set up DMA address ;BDDS entry point
0080 = 000D = 0024 = 0000	GNFDMA EQU GNFSVL EQU GNFFCL EQU GNFSV: DS	80H 13 36 GNFSVL	;Default DMA address ;Save length (no. of chars to move) ;File control block length ;Save area for file name/type
000D E5 000E D5 000F F5	GNF: PUSH PUSH PUSH	H D PSW	;Save FCB pointer ;Save file name pointer ;Save first/next flag
0010 118000 0013 0EIA 0015 CD0500 0018 F1 0019 E1	LXI MVI CALL POP POP	D,GNFDMA C,B\$SETDMA BDOS PSW H	;Set DMA to known address ;Function code ;Recover first/next flag ;Recover file name pointer
001A D1 001B D5	POP PUSH	D D	;Recover FCB pointer ;Resave FCB pointer
001C B7 001D C23E00 0020 CD9300 0023 E1 0024 D8 0025 E5	ORA JNZ CALL POP RC PUSH	A GNFN BF H	;Check if FIRST or NEXT ;NEXT ;Build file control block ;Recover FCB pointer (to balance stack) ;Return if error in file name ;Resave FCB pointer
0026 110000 0029 OEOD 0028 CD8A00 002E D1	LXI MVI CALL POP	D, GNFSV C, GNFSVL MOVE D	;Move ambiguous file name to ;save area ;HL -> FCB ;DE -> save area ;Get save length ;Recover FCB pointer
002F D5 0030 0E11	PUSH MVI	D C,B\$SEARCHF	;and resave ;Search FIRST
0032 CD0500 0035 E1 0036 FEFF 0038 CA7D00 003B C35D00	CALL POP CPI JZ JMP	BDOS H OFFH GNFEX GNFC	;Recover FCB pointer ;Check for error ;Error exit ;Common code
	GNFN:		<pre>;Execute search FIRST to re- ;establish contact with ;previous file ;User's FCB still has ;name/type in it</pre>
003E CD7F00 0041 D1 0042 D5 0043 0E11 0045 CD0500	CALL POP PUSH MVI CALL	GNFZF D D C,B\$SEARCHF BDOS	;Zero-fill all but file name/type ;Recover FCB address ;and resave ;Re-find the file
0048 D1 0049 D5 004A 210000	POP PUSH LXI	D D H,GNFSV	;Recover FCB pointer ;and resave ;Move file name from save area ;into FCB
004D 0E0D 004F CD8A00	MVI CALL	C,GNFSVL MOVE	;Save area length
0052 0E12 0054 CD0500 0057 E1 0058 FEFF 005A CA7D00	MVI CALL POP CPI JZ	C,B\$SEARCHN BDOS H OFFH GNFEX	;Search NEXT ;Recover FCB address ;Check for error ;Error exit
005D E5 005E 87	GNFC: PUSH ADD	H A	;Save FCB address ;Multiply BDOS return code * 32

Figure 5-17. (Continued)

005F 87	ADD	•	- × 4
003F 87	ADD	A	;* 4 ;* 8
0061 87	ADD	Å	;* 0 ;* 16
0062 87	ADD	A	;* 32
0063 218000	LXI	H, GNFDMA	;HL -> DMA address
0066 5F	MOV	E,A	;Make (code * 32) a word value
			;in DE
0067 1600	MVI	D, O	
0069 19	DAD	D	;HL →> file's directory entry
			;Move file name into FCB
006A D1	POP	D	;Recover FCB address
006B E5	PUSH	н	Save directory entry pointer
006C D5	PUSH	D	and resave
006D 0E0D 006F CD8A00	MVI CALL	C,GNFSVL MOVE	;Length of save area
0072 3A0000	LDA	GNESV	;Get disk from save area
0072 SHOODO	POP	D	Recover FCB address
0076 12	STAX	Ď	Overwrite user number in FCB
			,
			;Set up to zero-fill tail end
			;of FCB
0077 CD7F00	CALL	GNFZF	;Zero-fill
007A E1	POP	н	Recover directory entry
			;pointer
007B AF	XRA	A	;Clear carry
007C C9	RET		
	GNFEX:		
007D 37	STC		;Set carry to indicate error
007E C9	RET		your carry to indicate criss
	;This subrout	e zero fill ine zero-fills the d type in an FCB.	bytes that follow the
	;Entry parame : DE ->	ters file control bloc	k
	GNFZF:		
007E 210000	LXI	H, GNFSVL	;Bypass area that holds file name
007F 210D00 0082 19	DAD	D	;HL -> FCB + GNFSVL
0083 54	MOV	D, H	; DE -> FCB + GNFSVL
0084 5D	MOV	E,L	
0085 13	INX	D	;DE -> FCB + GNFSVL + 1
0086 3600	MVI	M, O	FCB + GNFSVL = 0
0088 0E17	MVI	C, GNFFCL-GNFSVL	;Remainder of file control block
	;Drop into MO	VE	
	;Spread O's t ;of FCB	nrough remainder	
	;MOVE		
		ine moves C bytes	from HL to DE.
	;This subrout	ine moves C bytes	from HL to DE.
	;This subrout MOVE:		
008A 7E	;This subrout MOVE: MOV	A, M	;Get source byte
008B 12	;This subrout MOVE: MOV STAX	A, M D	;Get source byte ;Save destination byte
008B 12	;This subrout MOVE: MOV STAX INX	A, M D D	;Get source byte ;Save destination byte ;Increment destination pointer
008B 12 008C 13 008D 23	;This subrout MOVE: MOV STAX	A, M D	;Get source byte ;Save destination byte
008B 12 008C 13 008D 23 008D 0D	;This subrout MOVE: STAX INX INX	A,M D D H	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count
008B 12 008C 13 008D 23	;This subrout MOVE: STAX INX DCR	A, M D D H C	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: STAX INX INX INX INX JNZ RET	A, M D D H C	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: INX INX INX DCR JNZ ;BF	A,M D H C MOVE	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: STAX INX INX INX INX JNZ RET	A,M D H C MOVE	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: STAX INX INX DCR JNZ RET ;BF ;Build file c ;This subrout	A,M D H H C MOVE Dontrol block ine formats a OOH-	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count ;Go back for more
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: STAX INX INX DCR JNZ RET ;BF ;Build file c ;This subrout ;(presumed to	A,M D D H C MOVE Dontrol block ine formats a OOH- be a file name) i	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count ;Go back for more byte terminated string .nto an FCB, setting the
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: MOV STAX INX INX DCR JNZ RET ;BF ;Build file c ;This subrout ;(presumed to ;disk and fil	A,M D D H C MOVE move ine formats a OOH- be a file name) i e name and type, a	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count ;Go back for more byte terminated string .nto an FCB, setting the
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: MOV STAX INX INX DCR JNZ RET ;BF ;Build file c ;This subrout ;(presumed to ;disk and fil	A,M D D H C MOVE Dontrol block ine formats a OOH- be a file name) i	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count ;Go back for more byte terminated string .nto an FCB, setting the
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: MOV STAX INX INX DCR JNZ RET ;BF ;Build file c ;This subrout ;(presumed to ;disk and fil	A,M D D H C MOVE move ine formats a OOH- be a file name) i e name and type, a	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count ;Go back for more byte terminated string .nto an FCB, setting the
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: MOV STAX INX INX DCR JNZ RET ;BF ;Build file c ;This subrout ;(presumed to ;disk and fil	A,M D D H C MOVE move ine formats a OOH- be a file name) i e name and type, a	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count ;Go back for more byte terminated string .nto an FCB, setting the
008B 12 008C 13 008D 23 008E 0D 008F C28A00	;This subrout MOVE: MOV STAX INX INX DCR JNZ RET ;BF ;Build file c ;This subrout ;(presumed to ;disk and fil	A,M D D H C MOVE move ine formats a OOH- be a file name) i e name and type, a	;Get source byte ;Save destination byte ;Increment destination pointer ;Increment source pointer ;Decrement count ;Go back for more byte terminated string .nto an FCB, setting the

Figure 5-17. (Continued)

```
;Entry parameters
; DE -> File control block (36 bytes)
; HL -> File name string (00H-byte-terminated)
;Exit parameters
; The built file control block
;This subroutine is shown in full in Figure 5-16
0093 C9 BF: RET ;Dummy subroutine for this example
```

Figure 5-17. (Continued)

Function 18: Search for Next Name Match

Function Code: C = 12HEntry Parameters: None (assumes previous Search First call) Exit Parameters: A = Directory code

Example

0012 = 0005 =	B\$SEARCHN BDOS	EQU EQU	18 5	;Search Next ;BDOS entry point
0000 0E12	MVI	C,B\$SEA	RCHN	;Function code ;Note: No FCB pointer ;You must precede this call : with a call to Search First
0002 CD0500	CALL	BDOS		;A = 0,1,2,3 ;(A * 32) + DMA -> directory ; entry ;A = OFFH if file name not ; found

Purpose

This function searches down the file directory for the *next* file name, type, and extent that match the FCB specified in a previous Search First function call.

Search First and Search Next are the only BDOS functions that must be used together. As you can see, the Search Next function does not require an FCB address as an input parameter—all the necessary information will have been left in the BDOS on the Search First call.

Like Search First, Search Next returns a directory code in the A register; in this case, if A = 0FFH, it means that there are no *more* files that match the file control block. If A is not 0FFH, it will be a value of 0, 1, 2, or 3, indicating the relative directory entry number.

Notes There are two ways of using the Search First/Next calls. Consider a simple file copying program that takes as input an ambiguous file name. You could scan the file directory, matching all of the possible file names, possibly displaying them on the console, and storing the names of the files to be copied in a table inside your program. This would have the advantage of enabling you to present the file names

to the operator before any copying occurred. You could even arrange for the operator to select which files to copy on a file-by-file basis. One disadvantage would be that you could not accurately predict how many files might be selected. On some hard disk systems you might have to accommodate several thousand file names.

The alternative way of handling the problem would be to match one file name, copy it, then match the next file name, copy it, and so on. If you gave the operator the choice of selecting which files to copy, this person would have to wait at the terminal as each file was being copied, but the program would not need to have large table areas set aside to hold file names. This solution to the problem is slightly more complicated, as you can see from the logic in Figure 5-17.

The subroutine in Figure 5-17, Get Next File (GNF), contains all of the necessary logic to search down a directory for both alternatives described. It does require that you indicate *on entry* whether it should search for the first or next file match, by setting A to zero or some nonzero value respectively.

You can see from Figure 5-17 that whenever the subroutine is called to get the *next* file, you must execute a Search First function to re-find the previous file. Only then can a Search Next be issued.

As with all functions that return a directory code in A, if this value is not 0FFH, it will be the relative directory entry number in the directory record currently in memory. This directory record will have been read into memory at whatever address was specified at the last Set DMA Address function call (code 26, 1AH). Notwithstanding its odd name, the DMA Address is simply the address into which any record input from disk will be placed. If the Set DMA Address function has not been used to change the value, then the CP/M default DMA address, location 0080H, will be used to hold the directory record.

The actual code for locating the address of the particular directory entry matched by the Search First/Next functions is shown in Figure 5-17 near the label GNFC. The method involves multiplying the directory code by 32 and then adding this product to the current DMA address.

Function 19: Erase (Delete) File

Function Code:C = 13HEntry Parameters:DE = Address of file control blockExit Parameters:A = Directory code

Example

0013	=	B\$ERASE	EQU	19	;Erase File
0005	=	BDOS	EQU	5	;BDOS entry point
		FCB:			;File control block
0000	00	FCB\$DISK:	DB	0	;Search on default disk drive
0001	3F3F4C454E	EFCB\$NAME:	DB	1??LENAN	1E' ;Ambiguous file name
0009	3F5950	FCB\$TYP:	DB	?YP ²	Ambiguous file type
0000	00	FCB\$EXTENT:	DB	0	Extent

	0000	FCB\$RESV:	DB	0,0	Reserved for CP/M
000F	00	FCB\$RECUSED:	DB	0	Records used in this extent;
0010	000000000	OFCB\$ABUSED:	DB	0,0,0,0	,0,0,0,0 ;Allocation blocks used
0018	000000000	0	DB	0,0,0,0	,0,0,0,0
0020	00	FCB\$SEQREC:	DB	0	;Sequential rec. to read/write
0021	0000	FCB\$RANREC:	DW	0	;Random rec. to read/write
0023	00	FCB\$RANRECO:	DB	0	;Random rec. overflow byte (MS)
0024	0E13	MVI	C,B\$ERA	ASE	;Function code
0026	110000	LXI	D,FCB		;DE -> file control block
0029	CD0500	CALL	BDOS		;A = OFFH if file not found

- **Purpose** This function logically deletes from the file directory files that match the FCB addressed by DE. It does so by replacing the first byte of each relevant directory entry (remember, a single file can have several entries, one for each extent) by the value 0E5H. This flags the directory entry as being available for use.
- **Notes** Like the previous two functions, Search First and Search Next, this function can take an ambiguous file name and type as part of the file control block, but unlike those functions, the logical disk select code cannot be a "?".

This function returns a directory code in A in the same way as the previous file operations.

Function 20: Read Sequential

Function Code:	C = 14H
Entry Parameters:	DE = Address of file control block
Exit Parameters:	A = Directory code

Example

0014 0005		B\$READSEQ BDOS	EQU EQU	20 5	;Read Sequential ;BDOS entry point
	00 464940454 545950	FCB: FCB\$DISK: EFCB\$NAME: FCB\$TYP:	DB DB DB DS	0 ^FILENA ^TYP^ 24	;File control block ;Search on default disk drive ME′ ;file name ;File type ;Set by file open
0026	0E14 110000 CD0500	MVI LXI CALL	C,B\$REA D,FCB BDOS	DSEQ	<pre>;Record will be read into ; address set by prior SETDMA ; call ;Function code ;DE -> File control block ;A = 00 if operation successful ;A = nonzero if no data in ; file</pre>

Purpose

This function reads the next record (128-byte sector) from the designated file into memory at the address set by the last Set DMA function call (code 26, 1AH). The record read is specified by the FCB's sequential record field (FCB\$SEQREC in the example listing for the Open File function, code 15). This field is incremented by 1 so that a subsequent call to Read Sequential will get the next record from the file. If the end of the current extent is reached, then the BDOS will

		lisk file. It ass	<t a<br="" character="" from="">sumes that the file has</t>			
	<pre>>>> Note : this subroutine changes CP/M's DMA address.</pre>					
	;Entry parame ; DE -:	eters ≻ file control bl	lock			
	;	ext character fr = OFFH on physic	cal end of file) rmal EOF character for			
	;Calling sequ ; LXI ; CALL ; CPI ; JZ ; CPI ; JZ	Dence DE,FCB GETC 1AH EOFCHAR OFFH ACTUALEOF				
0014 = 001A = 0005 =	B\$READSEQ B\$SETDMA BDOS	EQU 20 EQU 26 EQU 5	;Read sequential ;Set DMA address ;BDOS entry point			
0080 = 0000 0080 00	GETCBS EQU GETCBF: DS GETCCC: DB	128 GETCBS O	;Buffer size ;Declare buffer ;Char. count (initially ;"empty")			
0081 3A8000 0084 B7 0085 CA9900	GETC: LDA ORA JZ	GETCCC A GETCFB	;Check if buffer is empty ;Yes, fill buffer			
0088 3D 0089 328000	GETCRE: DCR STA	A GETCCC	;Re-entry point after buffer filled ;No, downdate count ;Save downdated count			
008C 47 008D 3E7F 008F 90 0090 5F 0091 1600 0093 210000 0096 19 0097 7E 0098 C9	MOV SUB MOV MVI LXI DAD RET	B,A A,GETCBS-1 B E,A D,O H,GETCBF D A,M	;Compute offset of next ;character ;By subtracting ;(buffer size downdated count) ;Make result into word value ;HL -> base of buffer ;HL -> next character in buffer ;Get next character			
0099 D5 009A 110000 009D 0E1A 009F CD0500 00A2 D1 00A3 0E14 00A5 CD0500 00A5 CD0500 00A8 B7 00A9 C2B400 00AC 3E80 00AE 328000 00AE C38800	GETCFB: PUSH LXI MVI CALL POP MVI CALL ORA JNZ MVI STA JMP	D D,GETCBF C,B\$SETDMA BDOS D C,B\$READSEQ BDOS A GETCX A,GETCBS GETCCC GETCRE	<pre>;Fill buffer ;Save FCB pointer ;Set DMA address to buffer ;function code ;Recover FCB pointer ;Read sequential "record" (sector) ;Check if read unsuccessful (A = NZ ;Yes ;Reset count ;Re-enter subroutine</pre>			
00B4 3EFF 00B6 C9	GETCX: MVI RET	A, OFFH	;Physical end of file ;Indicate such			

Figure 5-18. Read next character from sequential disk file

automatically open the next extent and reset the sequential record field to 0, ready for the next Read function call.

The file specified in the FCB must have been readied for input by issuing an Open File (code 15, 0FH) or a Create File (code 22, 16H) BDOS call.

The value 00H is returned in A to indicate a successful Read Sequential operation, while a nonzero value shows that the Read could not be completed because there was no data in the next record, as at the end of file.

Notes

Although it is not immediately obvious, you can change the sequential record number, FCB\$SEQREC, and within a given extent, read a record at random. If you want to access any given record within a file, you must compute which extent that record would be in and set the extent field in the file control block (FCB\$EX-TENT) before you open the file. Thus, although the function name implies sequential access, in practice you can use it to perform a simple type of random access. If you need to do true random access, look ahead to the Random Read function (code 33), which takes care of opening the correct extent automatically.

Figure 5-18 shows an example of a subroutine that returns the data from a sequential file byte-by-byte, reading in records from the file as necessary. This subroutine, GETC, is useful as a low-level "primitive" on which you can build more sophisticated functions, such as those that read a fixed number of characters or read characters up to a CARRIAGE RETURN/LINE FEED combination.

When you read data from a CP/M text file, the normal convention is to fill the last record of the file with 1AH characters (CONTROL-Z). Therefore, two possible conditions can indicate end-of-file: either encountering a 1AH, or receiving a return code from the BDOS function (in the A register) of 0FFH. However, if the file that you are reading is not an ASCII text file, then a 1AH character has no special meaning—it is just a normal data byte in the body of the file.

Function 21: Write Sequential

Function Code:C = 15HEntry Parameters:DE = Address of file control blockExit Parameters:A = Directory code

Example

0015 0005		B\$WRITES BDOS	EQ	EQU EQU	21 5	;Write Sequential ;BDOS entry point
	00 4649404548 545950	FCB: FCB\$DISK FCB\$NAME FCB\$TYP:	:	DB DB DB DS		;File control block ;Search on default disk drive Æ′ ;file name ;File type ;Set by Open or Create File
	0E15 110000 CD0500		MVI LXI CALL	C,B\$WRI1 D,FCB BDOS	reseq	;Record must be in address ; set by prior SETDMA call ;Function code ;DE -> File control block ;A = 00H if operation ; successful ;A = nonzero if disk full

Purpose This function writes a record from the address specified in the last Set DMA (code 26, 1AH) function call to the file defined in the FCB. The sequential record number in the FCB (FCB\$SEQREC) is updated by 1 so that the next call to Write Sequential will write to the next record position in the file. If necessary, a new extent will be opened to receive the new record.

This function is directly analogous to the Read Sequential function, writing instead of reading. The file specified in the FCB must first be activated by an Open File (code 15, 0FH) or create File call (code 22, 16H).

A directory code of 00H is returned in A to indicate that the Write was successful; a nonzero value is returned if the Write could not be completed because the disk was full.

As with the Read Sequential function (code 20, 14H), you can achieve a simple form of random writing to the file by manipulating the sequential record number (FCB\$SEQREC). However, you can only overwrite *existing* records in the file, and if you want to move to another extent, you must close the file and reopen it with the FCB\$EXTENT field set to the correct value. For true random writing to the file, look ahead to the Write Random function (code 34, 22H). This takes care of opening or creating the correct extent of the file automatically.

The only logical error condition that can occur when writing to a file is insufficient room on the disk to accommodate the next extent of the file. Any hardware errors detected will be handled by the disk driver built into the BIOS or BDOS.

Figure 5-19 shows a subroutine, PUTC, to which you can pass data a byte at a time. It assembles this data into a buffer, making a call to Write Sequential whenever the buffer becomes full. You can see that provision is made in the entry parameters (by setting register B to a nonzero value) for the subroutine to fill the remaining unused characters of the buffer with 1AH characters. You must do this to denote the end of an ASCII text file.

Function 22: Create (Make) File

Function Code:	C = 16H
Entry Parameters:	DE = Address of file control block
Exit Parameters:	A = Directory code

Example

Notes

0016 =	B\$CREATE	EQU	22 ;File Create	
0005 =	BDOS	EQU	57 ;BDOS entry point	
0000 00 0001 46494C454 0009 545950 000C 00	FCB: FCB\$DISK: HEFCB\$NAME: FCB\$TYP: FCB\$EXTENT:	DB DB DB DB	;File control block O ;Search on default disk 'FILENAME' ;file name 'TYP' ;File type O ;Extent	drive

٦

000F 0010	000000000	FCB\$RESV: FCB\$RECUSED: OFCB\$ABUSED:	DB DB DB		;Reserved for CP/M ;Records used in this extent ,0,0,0,0 ;Allocation blocks used
	0000000000	-	DB		,0,0,0,0
0020		FCB\$SEQREC:	DB	0	;Sequential rec. to read/write
	0000	FCB\$RANREC:	DW	0	;Random rec. to read/write
0023	00	FCB\$RANRECO:	DB	0	;Random rec. overflow byte (MS)
0026	0E16 110000 CD0500	MVI LXI CALL	C, B\$CRE D, FCB BDOS	ATE	;Note : file to be created ;must not already exist ;Function code ;DE -> file control block ;A = 0,1,2,3 if operation ; successful ;A = OFFH if directory full

		;to a s ;(128-b ;remain	equenti yte sec der of	al file, tors) or,	writin if re nt "re	the next chararacter out g out completed "records" quested to, will fill the cord" with IAH's to
		;Entry (; ; ;	DE -> B = 0,	File cont A = next	data	ock Sharacter to be output ent "record" with 1AH/s
		;Exit p	aramete none.	rs		
		;Callin	g seque	nce		
		; ; ; ; 01	LXI MVI LDA CALL	D,FCB B,O CHAR PUTC	;Not	end of file
		;;;;	LXI MVI CALL	D,FCB B,1 PUTC	;Indi	cate end of file
0015		B\$WRITE:		EQU	21	;Write sequential
001A 0005		B\$SETDM BDOS	A	EQU EQU	26 5	;Set DMA address ;BDOS entry point
0080 0000 0080		PUTCBS PUTCBF: PUTCCC:	DS	128 PUTCBS 0		;Buffer size ;Declare buffer ;Char. count (initially "empty")
		PUTC:				
0081			PUSH	D		;Save FCB address
0082 0083			PUSH MOV	PSW A,B		Save data character
0083			ORA	А, В А		;Check if end of file requested
	C29900		JNZ	PUTCEF		;Yes
0088	CDC300		CALL	PUTCGA		;No, get address of next free byte ;HL -> next free byte ;E = Current char. count (as
008B	F1		POP	PSW		;well as A) :Recover data character
0080			MOV	M.A		:Save in buffer
008D	7B		MOV	A,E		;Get current character count
008E			INR	Α		;Update character count
	FE80		CPI	PUTCBS		Check if buffer full
	CAA900 328000		JZ STA	PUTCWB PUTCCC		;Yes, write buffer ;No, save updated count
0097			POP	D		:Dump FCB address for return
0098			RET	-		,, . en eseren ier ierein



Г

		PUTCEF:			End of file
	0099 F1		POP	PSW	Dump data character
	009A CDC300		CALL	PUTCGA	HL -> next free byte
					A = current character count
1		PUTCCE:			:Copy EOF character
	009D FE80		CPI	PUTCBS	Check for end of buffer
	009F CAA900		JZ	PUTCWB	;Yes, write out the buffer
	00A2 361A		MVI	M,1AH	No, store EOF in buffer
	00A4 3C		INR	Α	;Update count
	00A5 23		INX	н	;Update buffer pointer
	00A6 C39D00		JMP	PUTCCE	;Continue until end of buffer
		PUTCWB:			;Write buffer
	00A9 AF		XRA	Α	Reset character count to 0
	00AA 328000		STA	PUTCCC	
	00AD 110000		LXI	D, PUTCBF	;DE -> buffer
	OOBO OE1A		MVI	C,B\$SETDMA	;Set DMA address -> buffer
	00B2 CD0500		CALL	BDOS	
	00B5 D1		POP	D	;Recover FCB address
	00B6 0E15		MVI	C,B\$WRITESEQ	;Write sequential record
	00B8 CD0500		CALL	BDOS	
1	OOBB B7		ORA	A	;Check if error
	00BC C2C000		JNZ	PUTCX	;Yes if A = NZ
	00BF C9		RET		;No, return to caller
1		PUTCX:			;Error exit
	OOCO 3EFF		MVI	A, OFFH	;Indicate such
	00C2 C9		RET		
		PUTCGA:			;Return with HL -> next free char.
					and A = current char. count
[00C3 3A8000		LDA	PUTCCC	:Get current character count
	00C6 5F		MOV	E,A	Make word value in DE
	00C7 1600		MVI	D, O	
1	0009 210000		LXI	H, PUTCBF	;HL -> Base of buffer
	00CC 19		DAD	D	;HL -> next free character
1	00CD C9		RET		
1					

Figure 5-19. Write next character to sequential disk file (continued)

Purpose This function creates a new file of the specified name and type. You must first ensure that no file of the same name and type already exists on the same logical disk, either by trying to open the file (if this succeeds, the file already exists) or by unconditionally erasing the file.

In addition to creating the file and its associated file directory entry, this function also effectively opens the file so that it is ready for records to be written to it.

This function returns a normal directory code if the file creation has completed successfully or a value of 0FFH if there is insufficient disk or directory space.

Notes

Under some circumstances, you may want to create a file that is slightly more "secure" than normal CP/M files. You can do this by using either lowercase letters or nongraphic ASCII characters such as ASCII NUL (00H) in the file name or type. Neither of these classes of characters can be generated from the keyboard; in the first case, the CCP changes all lowercase characters to uppercase, and in the second, it rejects names with odd characters in them. Thus, computer operators cannot erase such a file because there is no way that they can create the same file name from the CCP.

The converse is also true; the only way that you can erase these files is by using a program that *can* set the exact file name into an FCB and then issue an Erase File function call.

Note that this function cannot accept an ambiguous file name in the FCB.

Figure 5-20 shows a subroutine that creates a file only after it has erased any existing files of the same name.

Function 23: Rename File

Function Code:	C = 17H
Entry Parameters:	DE = Address of file control block
Exit Parameters:	A = Directory code

Example

0017		B\$RENAME	EQU	23	;Rename file
0005		BDOS	EQU	5	;BDOS entry point
0009	00 4F4C444E41 545950 00000000	FCB:	DB DB DB DB	0 10LDNAME 1TYP1 0,0,0,0	;File control block ;Search on default disk drive 5 ′ ;File name ;File type

		;CF ;Create file ;This subroutine creates a file. It erases any ;previous file before creating the new one.						
		:Entry parameters						
ĺ		;	DE -> F	ile cont	rol bloc	k for new file		
ļ		;Exit p	arameter	s				
		;	Carry c			n successful		
		;	Carry s		0, 1, 2, 3 ror (A =	OFFH)		
		•Callin	g sequen	.				
		;	LXI	D,FCB				
		;	CALL JC	CF ERROR				
	0013 =	B\$ERASE		EQU	19	;Erase file		
ļ	0016 =	B\$CREAT		EQU	22	;Create file		
	0005 =	BDOS		EQU	5	;BDOS entry point		
		CF:						
	0000 D5		PUSH	D		Preserve FCB pointer		
	0001 0E13 0003 CD0500		MVI CALL	C,B\$ERA BDOS	SE	;Erase any existing file		
	0006 D1		POP	D		Recover FCB pointer		
	0007 0E16 0009 CD0500		MVI CALL	C,B\$CRE BDOS	ATE	;Create (and open new file)		
	000C FEFF		CPI	OFFH		;Carry set if OK, clear if error		
	000E 3F 000F C9		CMC RET			;Complete to use Carry set if Error		
	000F 0.9							

Figure 5-20. Create file request

0010 00 0011 4E45574E41 0019 545950 001C 00000000			;FCB + 16 ME /;File name ;File type D
0020 0E17 0022 110000 0025 CD0500	MVI LXI CALL	C,B\$RENAME D,FCB BDOS	;Function code ;DE -> file control block ;A = OOH if operation succesful :A = OFFH if file not found

Purpose This function renames an existing file name and type to a new name and type. It is unusual in that it uses a single FCB to store both the old file name and type (in the first 16 bytes) and the new file name and type (in the second 16 bytes).

This function returns a normal directory code if the file rename was completed successfully or a value of 0FFH if the old file name could not be found.

Notes The Rename File function only checks that the old file name and type exist; it makes no check to ensure that the new name and type combination does not already exist. Therefore, you should try to open the new file name and type. If you succeed, do not attempt the rename operation. CP/M will create more than one file of the same name and type, and you stand to lose the information in both files as you attempt to sort out the problem.

For security, you can also use lowercase letters and nongraphic characters in the file name and type, as described under the File Create function (code 22, 16H) above.

Never use ambiguous file names in a rename operation; it produces strange effects and may result in files being irreparably damaged. This function will change *all* occurrences of the old file name to the new name.

Figure 5-21 shows a subroutine that will accept an existing file name and type and a new name and type and rename the old to the new. It checks to make sure that the new file name does not already exist, returning an error code if it does.

Function 24: Get Active Disks (Login Vector)

Function Code: C = 18H Entry Parameters: None Exit Parameters: HL = Active disk map (login vector)

Example

0018 =	B\$GETACTDSK	EQU	24	;Get Active Disks
0005 =	BDOS	EQU	5	;BDOS entry point
0000 0E18 0002 CD0500	MVI CALL	C,B\$GETA BDOS	ACTDSK	;Example of getting active ; disk function code ;HL = active disk bit map ;Bits are = 1 if disk active ;Bits 15 14 13 2 1 O ;Disk P O N C B A

Purpose This function returns a bit map, called the *login vector*, in register pair HL, indicating which logical disk drives have been selected since the last warm boot or

```
;RF
                   :Rename file
                   :This subroutine renames a file.
                   ;It uses the BF (build FCB) subroutine shown in Figure 5.16
                   ;Entry parameters
                            *** No case-folding of file names occurs ***
HL -> old file name (00-byte terminated)
                   ;
                            DE -> new file name (00-byte terminated)
                   :
                   ;Exit parameters
                            Carry clear if operation successful
(A = 0,1,2,3)
Carry set if error
                   $
                   :
                   :
                                      A = OFEH if new file name already exists
A = OFFH if old file name does not exist
                   :
                   :
                   ;Calling sequence
                            LXI
                                      H, OLDNAME
                                                         ;HL -> old name
;DE -> new name
                            LXI
                                      D, NEWNAME
                            CALL
                                      RF
                            JC.
                                      ERROR
000F =
                  B$OPEN
                                      EQU
                                               15
                                                         :Open file
0017 =
                  B$RENAME
                                      EQU
                                               23
5
                                                         ;Rename file
;BDOS entry point
0005 =
                  BDOS
                                      EQU
0000 000000000RFFCB:
0010 0000000000
                            DW
                                      0,0,0,0,0,0,0,0 ;1 1/2 FCB's long
                            DW
                                      0,0,0,0,0,0,0,0
0020 0000000000
                            DW
                                      0,0,0,0,0,0,0,0
0030 000000
                            DW
                                      0,0,0
                  RF:
0036 D5
                            PUSH
                                                         ;Save new name pointer
                                      n
0037 110000
                                      D.RFFCB
                            LXI
                                                         ;Build old name FCB
                                                         ;HL already -> old name
003A CD5D00
                            CALL
                                      BF
003D E1
                            POP
                                                         ;Recover new name pointer
003E 111000
                            LXI
                                      D, RFFCB+16
                                                         ;Build new name in second part of file
0041 CD5D00
                            CALL
                                      BF
                                                         ;control block
0044 111000
                            1 X 1
                                      D, RFFCB+16
                                                         ;Experimentally try
0047 0E0F
0049 CD0500
                                     C, B$OPEN
                            MUT
                                                         ;to open the new file
                           CALL
                                      BDOS
                                                         ;to engure it does
004C FEFF
                                      OFFH
                                                         not already exist
004E 3EFE
                            MVI
                                      A. OFEH
                                                         ;Assume error (flags unchanged)
;Carry set if A was 0,1,2,3
0050 D8
                            RC
                                     D, RFFCB
0051 110000
                            LXI
                                                         ;Rename the file
0054 0E17
                            MVI
                                     C, B$RENAME
0056 CD0500
                            CALL
                                      BDOS
0059 FEFF
                            CPI
                                      OFFH
                                                         ;Carry set if OK, clear if error
005B 3F
                            CMC
                                                         ;Invert to use carry, set if error
0050 09
                            RET
                  :BF
                  ;Build file control block
                  ;This subroutine formats a OOH-byte terminated string
                  ;(presumed to be a file name) into an FCB, setting the ;disk and the file name and type, and clearing the ;remainder of the FCB to 0's.
                  Entry parameters
                           DE -> file control block (36 bytes)
                  :
                            HL -> file name string (OOH-byte terminated)
                  :
                  ;Exit parameters
                            The built file control block.
                  ;Calling sequence
                           LXI
                                     D,FCB
                  ;
                            LXI
                                      H, FILENAME
                  ;
                            CALL
                                      RE
                  ;
                  BF:
005D C9
                            RET
                                                         ;Dummy subroutine : see Figure 5.16.
```

Figure 5-21. Rename file request

Reset Disk function (code 13, 0DH). The least significant bit of L corresponds to disk A, while the highest order bit in H maps disk P. The bit corresponding to the specific logical disk is set to 1 if the disk has been selected or to 0 if the disk is not currently on-line.

Logical disks can be selected programmatically through any file operation that sets the drive field to a nonzero value, through the Select Disk function (code 14, 0EH), or by the operator entering an "X:" command where "X" is equal to A, B, ..., P.

Notes This function is intended for programs that need to know which logical disks are currently active in the system—that is, those logical disks which have been selected.

Function 25: Get Current Default Disk

Function Code: C = 19HEntry Parameters: None Exit Parameters: A = Current disk(0 = A, 1 = B, ..., F = P)

Example

0019 =	B\$GETCURDSK	EQU 25	;Get Current Disk
0005 =	BDOS	EQU 5	;BDOS entry point
0000 0E19	MVI	C,B\$GETCURDSK	;Function code
0002 CD0500	CALL	BDOS	;A = 0 if A:, 1 if B:

Purpose This function returns the current default disk set by the last Select Disk function call (code 14, 0EH) or by the operator entering the "X:" command (where "X" is A, B, ..., P) to the CCP.

Notes This function returns the current default disk in coded form. Register A = 0 if drive A is the current drive, 1 if drive B, and so on. If you need to convert this to the corresponding ASCII character, simply add 41H to register A.

Use this function when you convert a file name and type in an FCB to an ASCII string in order to display it. If the first byte of the FCB is 00H, the current default drive is to be used. You must therefore use this function to determine the logical disk letter for the default drive.

Function 26: Set DMA (Read/Write) Address

Function Code: C = 1AHEntry Parameters: DE = DMA (read/write) address Exit Parameters: None

Example

001A =	B\$SETDMA	EQU	26	;Set DMA Address
0005 =	BDOS	EQU	5	;BDOS entry point

0000	SECBUFF:	DS	128	;Sector buffer
0080 0E1A 0082 110000 0085 CD0500	MVI LXI CALL	C,B\$SE D,SECB BDOS		;Function code ;Pointer to buffer

Purpose This function sets the BDOS's direct memory access (DMA) address to a new value. The name is an historic relic dating back to the Intel Development System on which CP/M was originally developed. This machine, by virtue of its hardware, could read data from a diskette directly into memory or write data to a diskette directly from memory. The name DMA address now applies to the address of the buffer to and from which data is transferred whenever a diskette Read, Write, or directory operation is performed.

Whenever CP/M first starts up (cold boot) or a warm boot or Reset Disk operation occurs, the DMA address is reset to its default value of 0080H.

Notes No function call can tell you the current value of the DMA address. All you can do is make a Set DMA function call to ensure that it is where you want it.

Once you have set the DMA address to the correct place for your program, it will remain set there until another Set DMA call, Reset Disk, or warm boot occurs.

The Read and Write Sequential and Random operations use the current setting of the DMA address, as do the directory operations Search First and Search Next.

Function 27: Get Allocation Vector

Function Code:C = 1BHEntry Parameters:NoneExit Parameters:HL = Address of allocation vector

Example

001B =	B\$GETALVEC	EQU 27	;Get Allocation Vector Address
0005 =	BDOS	EQU 5	;BDOS entry point
0000 0E1B 0002 CD0500	MVI CALL	C,B\$GETALVEC BDOS	;Function code ;HL -> Base address of ; allocation vector

- **Purpose** This function returns the base, or starting, address of the allocation vector for the currently selected logical disk. This information, indicating which parts of the disk are assigned, is used by utility programs and the BDOS itself to determine how much unused space is on the logical disk, to locate an unused allocation block in order to extend a file, or to relinquish an allocation block when a file is deleted.
- **Notes** Digital Research considers the actual layout of the allocation vector to be proprietary information.

Function 28: Set Logical Disk to Read-Only Status

Function Code: C = 1CHEntry Parameters: None Exit Parameters: None

Example

001C =	B\$SETDSKRO	EQU	28	:Set disk to Read Only ; function code
0005 =	BDOS	EQU	5	BDOS entry point
0000 0E1C 0002 CD0500	MVI Call	C.B\$SE BDOS	TDSKRO	:Sets disk selected by prior :Select disk function call :Function code

Purpose This function logically sets the currently selected disk to a Read-Only state. Any attempts to execute a Write Sequential or Write Random function to the selected disk will be intercepted by the BDOS, and the following message will appear on the console:

BDOS Err on X: R/O

where X: is the selected disk.

Notes Once you have requested Read-Only status for the currently selected logical disk, this status will persist even if you proceed to select other logical disks. In fact, it will remain in force until the next warm boot or Reset Disk System function call.

Digital Research documentation refers to this function code as Disk Write Protect. The Read-Only description is used here because it corresponds to the error message produced if your program attempts to write on the disk.

Function 29: Get Read-Only Disks

Function Code: C = 1DHEntry Parameters: None Exit Parameters: HL = Read-Only disk map

Example

001D =	B\$GETRODSKS	EQU 29	;Get Read Only disks
0005 =	BDOS	EQU 5	;BDOS entry poipt
0000 0E19 0002 CD0500	MVI CALL	C,B\$GETRODSKS BDOS	;Function code ;HL = Read Only disk bit map ;Bits are = 1 if disk Read Only ;Bits 15 14 13 2 1 O ;Disk P O N C B A

Purpose This function returns a bit map in registers H and L showing which logical disks in the system have been set to Read-Only status, either by the Set Logical

Disk to Read-Only function call (code 28, 1CH), or by the BDOS itself, because it detected that a diskette had been changed.

The least significant bit of L corresponds to logical disk A, while the most significant bit of H corresponds to disk P. The bit corresponding to the specific logical disk is set to 1 if the disk has been set to Read-Only status.

Function 30: Set File Attributes

Function Code:	C = 1EH
Entry Parameters:	DE = Address of FCB
Exit Parameters:	A = Directory code

Example

	B\$SETFAT BDOS	r	EQU EQU	30 5	;Set File Attribute ;BDOS entry point
	FCB:			_	File control block
00	FCB\$DIS	<:	DB	0	;Search on default disk drive
46494C454E	EFCB\$NAME	E:	DB	'FILENA	ME' ;File name
D4	FCB\$TYP:	:	DB	111+80H	;Type with R/O : attribute
5950			DB	/VP/	, attribute
	D C		DW	••	,0,0,0,0,0,0,0
OE1E		MVI	C,B\$SET	FAT	;Function code
110000		LXI	D,FCB		;DE -> file control block ;MS bits set in file name/type
CD0500		CALL	BDOS		A = OFFH if file not found
	D4 5950	= BDOS FCB: 00 FCB+DIS 46494C454EFCB\$NAME D4 FCB+TYP: 5950 0000000000 OE1E 110000	= BDOS FCB: 00 FCB#DISK: 46494C454EFCB#NAME: D4 FCB#TYP: 5950 0000000000 OE1E MVI 110000 LXI	= BDOS EQU FCB: FCB*DISK: DB 00 FCB*DISK: DB 45494C454EFCB*NAME: DB D4 FCB*TYP: DB 5950 DB 00000000000 DW 0E1E MVI C, B*SET 110000 LXI D, FCB	= BDOS EQU 5 FCB: 00 FCB\$DISK: DB 0 46494C454EFCB\$NAME: DB 'FILENA D4 FCB\$TYP: DB 'T'+80H 5950 DB 'YP' 0000000000 DW 0,0,0,0 OE1E MVI C,B\$SETFAT 110000 LXI D,FCB

Purpose

This function sets the bits that describe attributes of a file in the relevant directory entries for the specified file. Each file can be assigned up to 11 file attributes. Of these 11, two have predefined meanings, four others are available for you to use, and the remaining five are reserved for future use by CP/M.

Each attribute consists of a single bit. The most significant bit of each byte of the file name and type is used to store the attributes. The file attributes are known by a code consisting of the letter "f" (for file name) or "t" (for file type), followed by the number of the character position and a single quotation mark. For example, the Read-Only attribute is t1'.

The significance of the attributes is as follows:

- fl' to f4' Available for you to use
- f5' to f8' Reserved for future CP/M use
- t1' Read-Only File attribute
- t2' System File attribute
- t3' Reserved for future CP/M use

Attributes are set by presenting this function with an FCB in which the unambiguous file name has been preset with the most significant bits set appropriately. This function then searches the directory for a match and changes the matched entries to contain the attributes which have been set in the FCB.

The BDOS will intercept any attempt to write on a file that has the Read-Only attribute set. The DIR command in the CCP does not display any file with System status.

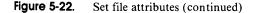
Notes You can use the four attributes available to you to set up a file security system, or perhaps to flag certain files that must be backed up to other disks. The Search First and Search Next functions allow you to view the complete file directory entry, so your programs can test the attributes easily.

The example subroutines in Figures 5-22 and 5-23 show how to set file attributes (SFA) and get file attributes (GFA), respectively. They both use a bit map in which the most significant 11 bits of the HL register pair are used to indicate the corresponding high bits of the 11 characters of the file name/type combination. You will also see some equates that have been declared to make it easier to manipulate the attributes in this bit map.

```
: SEA
                ;Set file attributes
                ;This subroutine takes a compressed bit map of all the
                ;file attribute bits, expands them into an existing
;file control block and then requests CP/M to set
                ; the attributes in the file directory.
                ;Entry parameters
                         DE -> file control block
                ;
                         HL = bit map. Only the most significant 11
                :
                               bits are used. These correspond directly with the possible attribute bytes.
                ;Exit parameters
                         Carry clear if operation successful (A = 0, 1, 2, 3)
                ;
                         Carry set if error (A = OFFH)
                ;
                ;Calling sequence
                                  D,FCB
                         LXI
                2
                         LXI
                                  H,0000$0000$1100$0000B ;Bit Map
                :
                         CALL
                                  SFA
                :
                         JC
                                  ERROR
                ;
                                                    ;File Attribute Equates
                                                             :F11 - F4
8000 =
                FA$F1
                         EQU
                                  1000$000$0000$00008
                                                             ;Available for use by
4000 =
                FA$F2
                         EQU
                                  0100$0000$0000$0000B
                                  0010$0000$0000$0000B
                                                             ; application programs
2000 =
                FA$E3
                         FOU
1000 =
                FA$F4
                         FOU
                                  0001$0000$0000$0000B
                                                             ;F51 - F81
                FA$F5
                                  0000$1000$0000$0000B
0800 =
                         EQU
0400 =
                                  0000$0100$0000$0000B
                                                             ;Reserved for CP/M
                FA$F6
                         EQU
                                  0000$0010$0000$0000B
0200 =
                FA$F7
                         EQU
0100 =
                FA$F8
                         EQU
                                  0000$0001$0000$0000B
                                                             :T1' -- read/only file
0080 =
                FA$T1
                         EQU
                                  0000$0000$1000$0000B
0080 =
                FA$R0
                         EQU
                                  FA$T1
0040 =
                FA$T2
                         EQU
                                  0000$0000$0100$0000B
                                                             :T2' -- system files
0040 =
                FA$SYS
                         FOU
                                  FA$T2
                                  0000$0000$0010$0000B
                                                             :T3' -- reserved for CP/M
0020 =
                FA$T3
                         EQU
001E =
                B$SETFAT
                                  EQU
                                           30
                                                    :Set file attributes
                BDOS
                                  EQU
                                           5
                                                    ;BDOS entry point
0005 =
```

Figure 5-22. Set file attributes

		SFA:			
0000	D5		PUSH	D	;Save FCB pointer
0001	13		INX	D	;HL -> 1st character of file name
0002	OEOB		MVI	C,8+3	;Loop count for file name and type
		SFAL:			;Main processing loop
0004	AF		XRA	Α	Clear carry and A
0005	29		DAD	н	;Shift next MS bit into carry
	CE00		ACI	0	;A = 0 or 1 depending on carry
0008			RRC		Rotate LS bit of A into MS bit;
0009			MOV	B,A	;Save result (OOH or 80H)
000A	EB		XCHG		;HL -> FCB character
000B	7E		MOV	A. M	:Get FCB character
0000	E67F		ANI	7FH	;Isolate all but attribute bit
000E	BO		ORA	В	;Set attribute with result
000F			MOV	M, A	;and store back into FCB
0010			XCHG		;DE -> FCB, HL = remaining bit map
0011			INX	D	;DE -> next character in FCB
0012			DCR	С	;Downdate character count
	C20400		JNZ	SFAL	;Loop back for next character
	OE1E		MVI	C,B\$SETFAT	;Set file attribute function code
0018			POP	D	Recover FCB pointer;
	CD0500		CALL	BDOS	
	FEFF		CPI	OFFH	;Carry set if OK, clear if error
001E			CMC		;Invert to use carry set if error
001F	C9		RET		



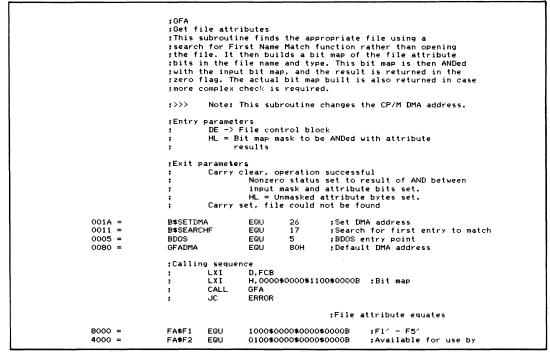


Figure 5-23. Get file attributes

0400 = Fr 0200 = Fr 0100 = Fr 0080 = Fr 0040 = Fr 0040 = Fr 0020 = Fr 0020 = Fr 0020 = Fr 0000 E5 0001 D5 0002 0E1A 0004 118000 0007 CD0500 000A D1 000B 0E11 000D CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0018 87 0018 87 0018 87 0018 87 0018 57 0018 57 0019 50 0022 23 0022 50 0022 50 0023 50 0023 50 0024 50 0025 50	A\$F5 EQU A\$F7 EQU A\$F7 EQU A\$F7 EQU A\$F7 EQU A\$F8 EQU A\$F1 EQU A\$70 EQU A\$70 EQU A\$72 EQU A\$73 EQU FA: PUSH PUSH PUSH PUSH VI LXI CALL CPI CMC JC	0000\$1000\$000 0000\$0100\$000 0000\$001\$000 0000\$0000\$100 FA\$T1 0000\$0000\$100 FA\$T2 0000\$0000\$010 FA\$T2 0000\$0000\$001 H D C.B\$SETDMA D,GFADMA BDOS	0\$0000B ;Reserved for CP/M 0\$0000B 0\$0000B ;T1 ⁷ read/only file 0\$0000B ;T2 ⁷ system files
0400 = Fr 0200 = Fr 0100 = Fr 0080 = Fr 0040 = Fr 0040 = Fr 0040 = Fr 0020 = Fr 0020 = Fr 0000 E5 0000 E5 0000 E5 0000 E10 0000 CD0500 0004 118000 0007 CD0500 0000 D0500 0000 D0500 0000 D0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0018 87 0018 87 0018 87 0018 87 0018 87 0018 5F 0012 189 0012 189 0012 199 0022 23 0023 EB 0024 0E0B 0026 210000 0027 CD050 0026 210000 0027 4A 0028 4680 0026 210000 0027 1A 0028 5F 0026 47 0029 1A 0028 5F 0026 47 0030 13 0031 0D 0035 29 0036 29 0037 29 0038 29 0038 29 0038 44 0032 47 0030 7B 0030 7B 0030 7B 0030 7B	A\$F6 EQU A\$F7 EQU A\$F8 EQU A\$F8 EQU A\$F7 EQU A\$F8 EQU A\$F7 EQU A\$F7 EQU A\$F2 EQU A\$F3 EQU A\$F3 EQU FA: PUSH MVI LXI CALL POP MVI CALL CPI CACL	0000\$0100\$000 0000\$0010\$000 0000\$0000\$0	0\$0000B ;Reserved for CP/M 0\$0000B 0\$0000B ;T1' read/only file 0\$0000B ;T2' system files 0\$0000B ;T3' reserved for CP/M ;Save AND-mask ;Save FCB pointer ;Set DMA to default address
0200 = Fr 0100 = Fr 0080 = Fr 0040 = Fr 0040 = Fr 0020 = Fr 0010 = Fr 0012 = Fr 0012 = Fr 0012 = Fr 0013 = DA4100 0014 = 87 0015 = Fr 0016 = 218000 0021 = 19 0022 = 23 0023 EB 0024 = 0024 E 680 0025 P 0031 0D 0	A\$F7 EQU FA\$F8 EQU FA\$F8 EQU FA\$F8 EQU FA\$F0 EQU FA\$F3 EQU FA\$T3 EQU FA: PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH PUSH CALL CPI CALL CPI CMC	0000\$00108000 0000\$00019000 FA\$T1 0000\$0000\$100 FA\$T2 0000\$0000\$001 H D C, B\$SETDMA D, GFADMA	O\$0000B 0\$0000B ;T1' read/only file 0\$0000B ;T2' system files 0\$0000B ;T3' reserved for CP/M ;Save AND-mask ;Save FCB pointer ;Set DMA to default address
0100 = Fr 0080 = Fr 0040 = Fr 0040 = Fr 0020 = Fr 0020 = Fr 0020 E15 0000 E5 0000 E5 0000 CD500 0007 CD0500 0004 118000 0007 CD0500 0000 CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0018 87 0018 87 0018 87 0018 5F 0012 1800 0021 19 0022 23 0023 EB 0024 0E0B 0026 210000 0021 19 0022 23 0023 EB 0024 0E0B 0026 210000 0021 9 0022 6F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 44 0036 47 0038 A4 0036 47 0038 A4 0036 47 0037 B5	FA\$F8 EQU FA\$T1 EQU FA\$T0 EQU FA\$T2 EQU FA\$T3 EQU FA: PUSH PUSH MVI LXI CALL POP MVI CALL CPI CPI CMC	0000\$0001\$000 FA\$T1 0000\$0000\$010 FA\$T2 0000\$0000\$001 H D C.B\$SETDMA D,GFADMA	0\$0000B 0\$0000B ;T1' read/only file 0\$0000B ;T2' system files 0\$0000B ;T3' reserved for CP/M :Save AND-mask ;Save FCB pointer ;Set DMA to default address
0080 = Fr 0040 = Fr 0040 = Fr 0020 = Fr 0020 = GF 0000 E5 0001 D5 0002 0E1A 0004 118000 0007 CD0500 000A D1 000B 0E11 000D CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0013 B7 0018 87 0018 87 0018 87 0018 87 0018 5F 0012 19 0022 23 0023 EB 0024 0E0B 0024 E680 0025 19 0022 EB 0024 E680 0027 19 0022 EB 0024 E680 0025 29 0031 0D 0035 29 0036 29 0037 29 0038 29 0038 29 0038 44 0032 47 0035 78 0035 78 0036 47 0037 78 0038 44 0032 47 0036 78 0036 78 0036 78 0037 78 0038 44 0036 47 0036 78 0036 78 0037 78 0038 47 0038 78 0037 78 0038 44 0036 47 0036 78 0036 78 0037 78 0037 78 0038 78 0038 78 0038 78 0038 78 0038 78 0038 78 0038 78 0038 78 0037 78 0038 78 0038 78 0037 78 0038 78 0038 78 0037 78 0038 78 0038 78 0037 78 0038 78	A\$T1 EQU A\$T0 EQU A\$T2 EQU A\$T3 EQU A\$T3 EQU FA: PUSH MVI LXI CALL CPI CALL CPI CMC	FA\$T1 0000\$0000\$010 FA\$T2 0000\$0000\$001 H D C.B\$SETDMA D,GFADMA	0\$0000B ;T2' system files 0\$0000B ;T3' reserved for CP/M ;Save AND-mask ;Save FCB pointer ;Set DMA to default address
0080 = F/ 0040 = F/ 0040 = F/ 0020 = F/ 0020 = F/ 0020 = F/ 0001 D5 0002 0E1A 0002 0E1A 0004 118000 0007 CD0500 0008 0E11 0000 CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0013 B7 0018 87 0018 87 0018 87 0018 87 0018 5F 0012 1800 0016 1600 0012 19 0022 23 0023 EB 0024 0E0B 0026 210000 0027 1A 0026 210000 0027 1A 0028 5F 0026 210000 0027 29 0026 4F 0027 29 0026 4F 0027 29 0030 13 0031 0D 0032 C2900 0035 29 0036 29 0038 29 0038 44 0036 47 0036 45 0036 45	A\$RO EQU A\$T2 EQU A\$T3 EQU A\$T3 EQU FA: PUSH MVI LXI CALL POP MVI CALL CPI CPI CMC	FA\$T1 0000\$0000\$010 FA\$T2 0000\$0000\$001 H D C.B\$SETDMA D,GFADMA	0\$0000B ;T2' system files 0\$0000B ;T3' reserved for CP/M ;Save AND-mask ;Save FCB pointer ;Set DMA to default address
0040 = Fr 0020 = Fr 0020 = Fr 0000 E5 0001 D5 0002 0E1A 0004 118000 0007 CD0500 000A D1 000B 0E11 000D CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0013 B7 0018 87 0018 87 0018 87 0018 87 0018 87 0018 5F 0012 19 0022 23 0023 EB 0024 0E0B 0024 0E0B 0025 210000 0021 19 0022 EB 0024 0E0B 0025 EB 0024 0E0B 0025 210000 0021 19 0022 EB 0024 0E0B 0025 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 A4 0032 47 0038 A5	A\$T2 EQU A\$T3 EQU A\$T3 EQU FA: PUSH PUSH PUSH VI LXI CALL CPI CALL CPI CMC	0000\$0000\$010 FA\$T2 0000\$0000\$001 H D C,B\$SETDMA D,GFADMA	0\$0000B ;T3' reserved for CP/M ;Save AND-mask ;Save FCB pointer ;Set DMA to default address
0020 = F/ 0000 E5 0001 D5 0002 0E1A 0004 118000 0007 CD0500 0008 0E11 0008 0E11 0008 0E11 0010 FEFF 0012 3F 0013 DA4100 0016 87 0018 87 0018 87 0018 87 0018 87 0018 5F 0011 1600 0016 1600 0016 218000 0016 218000 0012 19 0022 23 0023 EB 0024 0E0B 0025 210000 0025 29 0036 21 0037 29 0036 29 0038 29 0038 29 0038 44 0036 47 0036 45 0036 45	FAST3 EQU FA: PUSH MVI LXI CALL CALL CPI CPI CMC	0000\$0000\$001 H D C,B\$SETDMA D,GFADMA	;Save AND-mask ;Save FCB pointer ;Sat DMA to default address
GF 0000 E5 0001 D5 0002 0E1A 0004 118000 0007 CD0500 000A D1 000B 0E11 000B 0E11 000B 0C0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0018 87 0018 87 0018 5F 0012 1600 0012 19 0022 23 0023 EB 0024 0E0B 0024 0E0B 0024 0E0B 0024 210000 0012 5F 0022 1A 0022 EB 0024 0E0B 0024 0E0B 0024 0E0B 0024 0E0B 0025 29 0030 13 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 A4 0032 47 0038 A5	FA: PUSH MVI LXI CALL POP MVI CALL CPI CMC	H D C,B\$SETDMA D,GFADMA	;Save AND-mask ;Save FCB pointer ;Sat DMA to default address
0000 E5 0001 D5 0002 0E1A 0004 118000 0007 CD0500 000B 0E11 000B 0E11 000B 0E11 0010 FEFF 0012 3F 0013 DA4100 0016 87 0018 87 0018 87 0018 87 0018 87 0019 87 0018 5F 0011 1600 0011 218000 0011 218000 0011 218000 0012 19 0022 23 0023 EB 0024 0E0B 0025 210000 0025 29 0036 21 0037 29 0036 29 0037 29 0038 29 0038 47 0038 47 0038 47 0038 47 0038 47 0038 47 0038 47 0036 45	PUSH PUSH MVI LXI CALL POP MVI CALL CPI CMC	D C,B\$SETDMA D,GFADMA	;Save FCB pointer ;Set DMA to default address
0001 D5 0002 0E1A 0004 118000 0007 CD0500 0008 DE11 000B 0E11 000B CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0018 87 0018 87 0018 87 0018 87 0018 87 0018 5F 0012 1600 0016 1600 0012 19 0012 218000 0021 19 0022 23 0023 EB 0024 0E0B 0022 23 0023 10000 0022 5P 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0038 29 0038 29 0038 44 0032 47 0038 A4 0032 47 0038 7B	PUSH MVI LXI CALL POP MVI CALL CPI CMC	D C,B\$SETDMA D,GFADMA	;Save FCB pointer ;Set DMA to default address
0002 0E1A 0004 118000 0007 CD0500 000A D1 000B 0E11 000D CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0018 87 0018 87 0018 87 0018 5F 001C 1600 0021 19 0022 23 0023 EB 0024 0E0B 0022 23 0023 EB 0024 0E0B 0022 23 0023 EB 0024 0E0B 0026 210000 0021 19 0022 23 0023 EB 0024 0E0B 0026 210000 0021 19 0022 6F 0027 29 0026 4F 0027 29 0030 13 0031 0D 0032 C2990 0035 29 0036 29 0037 29 0036 29 0038 29 0038 44 0032 47 0039 D1 0038 A4 0032 47 0039 7B	MVI LXI CALL POP MVI CALL CPI CMC	C,B\$SETDMA D,GFADMA	;Set DMA to default address
0004 118000 0007 CD0500 000B 0E11 000D CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0018 87 0019 87 0018 87 0019 87 0019 87 0018 5F 0011 1400 0011 218000 0011 218000 0011 218000 0011 218000 0011 218000 0011 218000 0011 218000 0011 218000 0011 218000 0011 218000 0012 19 0022 23 0023 EB 0024 060B 0026 210000 0025 29 0026 4F 0027 29 0030 10 0031 00 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0038 47 0038 47 0038 45	LXI CALL POP MVI CALL CPI CMC	D,GFADMA	
0007 CD0500 000A D1 000B 0E11 000B CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0018 87 0018 87 0018 87 0018 87 0018 87 0018 5F 0011 1600 0011 1600 0012 18000 0022 180 0022 23 0023 EB 0024 0E0B 0022 23 0023 EB 0024 0E0B 0022 23 0023 EB 0024 0E0B 0022 6F 0027 1A 0028 5F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0038 29 0038 29 0039 D1 0038 A4 0032 47 0038 A5	CALL POP MVI CALL CPI CMC		
000A D1 000B 0E11 000D CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0018 87 0019 87 0019 87 0019 87 0019 87 0019 87 0012 12000 0021 19 0022 23 0023 EB 0024 0E0B 0026 210000 0027 1A 0026 210000 0026 210000 0027 29 0026 45 0027 29 0030 13 0031 0D 0032 C2900 0035 29 0036 29 0036 29 0038 29 0038 29 0038 44 0036 47 0036 45	POP MVI CALL CPI CMC	BDOS	FUE TY DAM BUUTESS
000B 0E11 000B CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0018 87 0018 87 0018 87 0018 5F 0010 1600 0021 19 0022 23 0023 EB 0024 0E08 0022 23 0023 EB 0024 0E08 0022 23 0023 EB 0024 0E08 0022 210000 0025 29 0026 4F 0027 29 0026 4F 0027 29 0031 0D 0032 C2900 0035 29 0036 29 0036 29 0037 29 0038 29 0038 29 0038 44 0032 47 0038 A4 0032 47 0038 A5	MVI CALL CPI CMC		
0000 CD0500 0010 FEFF 0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0019 87 0019 87 0018 5F 0011 1600 0016 1600 0012 119 0022 23 0023 EB 0024 0E0B 0025 210000 0025 29 0026 6F 0027 4 0027 6F 0027 29 0030 13 0031 0D 0032 59 0036 29 0037 29 0036 29 0037 29 0038 29 0038 29 0038 44 0032 47 0032 7B	CALL CPI CMC	D	;Recover FCB pointer
0010 FEFF 0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0018 87 0018 87 0018 5F 0016 1600 0011 1600 0011 218000 0021 19 0022 23 0023 EB 0024 0E0B 0022 EB 0024 0E0B 0022 210000 0029 1A 0020 F 0029 1A 0020 E5 0022 6F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0038 29 0038 29 0038 29 0038 29 0038 47 0038 A4 0032 47 0036 A5	CPI CMC	C, B\$SEARCHF	;Search for match with name
0012 3F 0013 DA4100 0016 87 0017 87 0018 87 0019 87 0018 87 0018 5F 0010 19 0022 23 0023 EB 0024 0E0B 0022 210000 0025 210000 0026 4F 0029 1A 0026 4F 0027 97 0028 5F 0027 29 0030 13 0031 0D 0032 22900 0035 29 0036 29 0036 29 0037 29 0038 29 0038 29 0038 29 0038 29 0038 29 0038 29 0038 47 0038 A4 0032 47 0038 A5	CMC	BDOS	
0013 DA4100 0016 87 0017 87 0018 87 0019 87 0018 57 0010 5F 0010 1600 0011 218000 0021 19 0022 23 0023 EB 0024 060B 0025 210000 0027 10 0027 10 0028 65 0026 6F 0027 29 0030 13 0031 00 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0038 29 0038 29 0038 47 0038 47 0038 A4 0036 45		OFFH	;Carry set if OK, clear if error
0016 87 0017 87 0018 87 0018 87 0018 87 0018 5F 0011 1600 0021 19 0022 23 0023 EB 0024 0E0B 0026 210000 0027 1A 0028 E80 0026 210000 0029 1A 0029 1A 0020 07 0020 55 0022 6F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0038 29 0038 29 0038 29 0038 47 0032 47 0032 7B 0032 45	JC		;Invert to use set carry if error
0017 87 0018 87 0019 87 0019 87 0011 5F 0011 1600 0011 218000 0021 19 0022 23 0023 EB 0024 0E0B 0025 210000 0027 10 0029 1A 0024 E680 0022 07 0022 6F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0038 29 0038 47 0038 A4 0032 47 0038 A5		GFAX	Return if error
0017 87 0018 87 0019 87 0019 87 0011 5F 0011 1600 0011 218000 0021 19 0022 23 0023 EB 0024 0E0B 0025 210000 0027 1A 0028 4680 0020 07 0020 85 0022 6F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0038 47 0038 A4 0032 47 0038 A5			;Multiply by 32 to get offset into DMA but
0017 87 0018 87 0019 87 0018 57 0011 5F 0010 1600 0010 1400 0011 218000 0021 19 0022 23 0023 EB 0024 0E0B 0026 210000 0027 1A 0028 4680 0020 07 0020 85 0022 6F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0035 29 0036 29 0037 29 0038 29 0038 29 0038 47 0038 A4 0032 47 0038 A5	ADD	A	;* 2
0019 87 001A 87 001B 5F 001C 1400 001E 218000 0021 19 0022 23 0023 EB 0024 0E0B 0025 210000 0025 210000 0026 4 0027 07 0020 B5 0022 6F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0037 29 0036 29 0038 29 0038 47 0038 A4 0032 47 0038 A5	ADD	Ą	;* 4
001A 87 001B 5F 001C 1600 002E 218000 0022 19 0022 23 0023 EB 0024 0E0B 0026 210000 0029 1A 0029 1A 0029 1A 0029 1A 0020 07 0020 07 0020 85 0022 6F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0038 29 0038 29 0038 47 0038 A4 0032 47 0038 A5	ADD	A	;* 8
0018 5F 0012 1600 0012 19 0022 23 0023 EB 0024 0E0B 0026 210000 0029 1A 0029 1A 0029 1A 0020 6F 0022 07 0020 B5 0022 6F 0027 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0037 29 0038 29 0038 29 0038 29 0038 29 0038 29 0038 47 0038 A4 0032 47 0038 7B	ADD	A	;* 16
001C 1600 001E 218000 0021 19 0022 23 0023 EB 0024 0E0B 0026 210000 0027 1A 0028 E680 002C 07 0020 B5 0022 6F 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0038 29 0038 29	ADD	A .	;* 32
001E 218000 0021 19 0022 23 0023 EB 0024 0E0B 0026 210000 0027 1A 002A E680 002C 07 002D B5 002E 6F 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 A4 0036 47 0030 7B 0036 A5	MOV	E,A	;Make into a word value
0021 19 0022 23 0023 EB 0024 0E0B 0026 210000 0029 1A 0029 1A 0020 07 0020 B5 002E 6F 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0037 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 7A 0038 A4 003C 47 0038 A5	MVI	D,O	
0022 23 0023 EB 0024 0E0B 0026 210000 0027 1A 0028 E680 002C 07 0020 B5 002E 6F 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0038 29 0038 29 0038 29		H,GFADMA	;HL -> DMA address ;HL -> Directory entry in DMA buffer
0023 EB 0024 0E0B 0026 210000 0027 1A 002A E480 002C 07 002D B5 002E 6F 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 A4 003C 47 003B A4 003C 47 003B A5	DAD INX	D H	;HL -> Directory entry in DMA buffer ;HL -> 1st character of file name
0026 210000 GF 0029 1A 002A E660 002C 07 002D B5 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0036 29 0037 29 0038 29 0039 D1 0038 A4 003C 47 003B A4 003C 47 003D 7B 003E A5	XCHG	п	;DE -> 1st character of file name
GF 0029 1A 002A E&80 002C 07 002D B5 002E 6F 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 A4 0032 47 0035 7B 0035 A5	MVI LXI	C,8+3 H,0	;Count of characters in file name and type ;Clear bit map
0029 1A 002A E680 002C 07 002D B5 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0037 29 0038 29 0039 D1 0038 A4 0038 A4 0038 A4 0032 47 0039 7B			
002A E680 002C 07 002D B5 002E 6F 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 7A 0038 A4 003C 47 003D 7B 003E A5	FAL:	D	;Main loop :Get next character of file name
002C 07 002D B5 002E 6F 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0038 29 0038 29 0038 29 0039 D1 003A 7A 003B A4 003C 47 003D 7B 003E A5	LDAX		;Get next character of file name
002D B5 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0036 29 0038 29 0039 D1 0034 7A 0038 A4 003C 47 003D 7B 0032 A5	ANI	80H	;Isolate attribute bit .Mova MS bit into IS bit
002E 6F 002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 7A 0038 A4 0032 47 0030 7B 0032 A5	RLC	1	;Move MS bit into LS bit ;OR in any previously set bits
002F 29 0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 A4 003C 47 003D 7B 003E A5	MOV	-	;uk in any previously set bits ;Save result
0030 13 0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0039 D1 0034 7A 0038 A4 003C 47 003D 7B 003E A5	DAD	L,A H	;Save result ;Shift HL left one bit for next time
0031 0D 0032 C22900 0035 29 0036 29 0037 29 0038 29 0039 D1 0034 7A 0038 A4 003C 47 003D 7B 003E A5	INX	H D	;DE -> next character in file name, type
0032 C22900 0035 29 0036 29 0037 29 0038 29 0038 29 0039 D1 0038 A4 003C 47 003D 7B 003E A5	DCR	C U	;Downdate count
0036 29 0037 29 0038 29 0039 D1 003A 7A 003B A4 003C 47 003D 7B 003E A5	JNZ	GFAL	;Go back for next character
0036 29 0037 29 0038 29 0039 D1 003A 7A 003B A4 003C 47 003D 7B 003E A5	DAD	н	;Left justify attribute bits in HL
0037 29 0038 29 0039 D1 003A 7A 003B A4 003C 47 003D 7B 003D 7B	DAD	H	;MS attribute bit will already be in
0038 29 0039 D1 003A 7A 003B A4 003C 47 003D 7B 003E A5	DAD	H	;bit 11 of HL, so only 4 shifts are
003A 7A 003B A4 003C 47 003D 7B 003E A5	DAD	н	;necessary
003A 7A 003B A4 003C 47 003D 7B 003E A5	POP	D	;Recover AND-mask
003B A4 003C 47 003D 7B 003E A5	MOV	Ã, D	;Get MS byte of mask
003C 47 003D 7B 003E A5	ANA	н	;AND with MS byte of result
003D 7B 003E A5	MOV	B,A	;Save interim result
003E A5	MOV	A,E	;Get LS byte of mask
	ANA	L	;AND with LS byte of result
	ORA	В	;Combine two results to set Z flag
0040 C9	RET		
	FAX:		Error exit
0041 E1	POP	н	;Balance stack
0042 C9	RET		

Figure 5-23. Get file attributes (continued)

Function 31: Get Disk Parameter Block Address

	Entry Para	Code: $C = 1F$ meters: None meters: $HL = Ac$			
Example	001F = 0005 =	B\$GETDPB BDOS	EQU EQU	31 5	;Get Disk Parameter Block ; Address ;BDOS entry point
	0000 0E1F 0002 CD0500	MVI CALL	C,B\$GET BDOS	IDPB	;Returns.DPB address of ; logical disk previously ; selected with a Select ; Disk function. ;Function code ;HL -> Base address of current ; disk's parameter block

Purpose This function returns the address of the disk parameter block (DPB) for the last selected logical disk. The DPB, explained in Chapter 3, describes the physical characteristics of a specific logical disk—information mainly of interest for system utility programs.

Notes

The subroutines shown in Figure 5-24 deal with two major problems. First, given a track and sector number, what allocation block will they fall into? Converseley, given an allocation block, what is its starting track and sector?

These subroutines are normally used by system utilities. They first get the DPB address using this BDOS function. Then they switch to using direct BIOS calls to perform their other functions, such as selecting disks, tracks, and sectors and reading and writing the disk.

The first subroutine, GTAS (Get Track and Sector), in Figure 5-24, takes an allocation block number and converts it to give you the starting track and sector number. GMTAS (Get Maximum Track and Sector) returns the maximum track and sector number for the specified disk. GDTAS (Get Directory Track and Sector) tells you not only the starting track and sector for the file directory, but also the number of 128-byte sectors in the directory.

Note that whenever a track number is used as an entry or an exit parameter, it is an absolute track number. That is, the number of reserved tracks on the disk before the directory has already been added to it.

GNTAS (Get Next Track and Sector) helps you read sectors sequentially. It adds 1 to the sector number, and when you reach the end of a track, updates the track number by 1 and resets the sector number to 1.

GAB (Get Allocation Block) is the converse of GTAS (Get Track and Sector). It returns the allocation block number, given a track and sector.

Finally, Figure 5-24 includes several useful 16-bit subroutines to divide the HL register pair by DE (DIVHL), to multiply HL by DE (MULHL), to subtract DE from HL (SUBHL —this can also be used as a 16-bit compare), and to shift HL right one bit (SHLR). The divide and multiply subroutines are somewhat primitive, using iterative subtraction and addition, respectively. Nevertheless, they do perform their role as supporting subroutines.

;Useful subroutines for accessing the data in the ;disk parameter block 000E = 14 B\$SELDSK FOU ;Select Disk function code 001F =R\$GETDPR FOU 31 :Get DPB address 0005 = BDOS FOU 5 BDDS entry point ;It makes for easier, more compact code to copy the specific disk parameter block into local variables ;while manipulating the information. ;Here are those variables --DPB: ;Disk parameter block 0000 0000 DPBSPT: DW ;128-byte sectors per track 0 0002 00 DPBBS: DB ;Block shift 0 0003 00 DPBBM: DB 0 ;Block mask ō :Extent mask DPREM. **DB** 0005 0000 DPRMAR: DW 0 :Maximum allocation block number DPBNOD: DW 0007 0000 0 ;Number of directory entries - 1 DPBDAB: DW ō Directory allocation blocks 000B 0000 DPBCBS: DW :Check buffer size 0 000D 0000 ;Tracks before directory (reserved tracks) DEBIED: DW 0 0.00F =DPBSZ EQU \$-DPB ;Disk parameter block size ; GETDPB ;Gets disk parameter block This subroutine copies the DPB for the specified flogical disk into the local DPB variables above. ;Entry parameters A = Logical disk number (A: = 0, B: = 1...) . ;Exit parameters Local variables contain DPB . GETDPB: 000F 5F MOV E,A ;Get disk code for select disk 0010 0E0E 0012 CD0500 C, B\$SELDSK MVI ;Select the disk CALL BDOS C, B\$GETDPB 0015 0E1F MVI ;Get the disk parameter base address 0017 CD0500 CALL BDOS ;HL -> DPB 001A 0E0F 001C 110000 ;Set count MVI C,DPBSZ ;Get base address of local variables 1 1 1 DDPR GDPBL: ;Copy DPB into local variables ;Get byte from DPB 001F 7E MOV A.M ;Store into local variable 0020 12 STAX D b ;Update local variable pointer ;Update DPB pointer 0021 13 0022 23 INX INX Ĥ 0023 OD DCR C ;Downdate count 0024 C21F00 0027 C9 GDPBL ;Loop back for next byte JNZ RET : GTAS ;Get track and sector (given allocation block number) ;This subroutine converts an allocation block into a strack and sector number -- note that this is based on ;128-byte sectors. ;>>>> Note: You must call GETDPB before
;>>>>> you call this subroutine ;Entry parameters HL = allocation block number 2 Exit parameters HL = track number . DE = sector number . :Method : ;In mathematical terms, the track can be derived from: ;Trk = ((allocation block * sec. per all. block) / sec. per trk)
; + tracks before directory

Figure 5-24. Accessing disk parameter block data

```
The sector is derived from:
                   ;Sec
                         -
                           ((allocation block * sec. per all. block) modulo/
                             sec. per trk) + 1
                   •
                   GTAS:
0028 340200
                             LDA
                                       DPBBS
                                                            :Get block shift -- this will be 3 to
                                                           ;7 depending on allocation block size
;It will be used as a count for shifting
                   GTASS:
002B 29
                             DAD
                                       н
                                                            ;Shift allocation block left one place
002C 3D
                             DCR
                                                            ;Decrement block shift count
002D C22B00
                             JNZ
                                       GTASS
                                                           ;More shifts required
;DE = all. block * sec. per block
0030 EB
                             XCHG
                                                            ; i.e. DE = total number of sectors
0031 2A0000
0034 EB
                             LHLD
                                       DPBSPT
                                                            ;Get sectors per track
                                                           ;HL = sec. per trk, DE = tot. no. of sec.
;BC = HL/DE, HL = remainder
;BC = track, HL = sector
                             XCHG
0035 CD8F00
                             CALL
                                       DIVHL
0038 23
                             INX
                                                            ;Sector numbering starts from 1
                                       н
0039 EB
003A 2A0D00
                                                            ;DE = sector, HL = track
                             XCHG
                             LHLD
                                       DPBTBD
                                                            ;Tracks before directory
003D 09
                             DAD
                                       в
                                                            ;DE = sector, HL = absolute track
003E C9
                             RET
                   ; GMTAS
                   ;Get maximum track and sector
                   ;This is just a call to GTAS with the maximum.
                   ;allocation block as the input parameter
                   ;>>>>> Note: You must call GETDPB before
                                    you call this subroutine
                   :>>>>>
                   ;Entry parameters: none
                   ;Exit parameters:
                             HL = maximum track number
                             DE = maximum sector
                   ;
                   GMTAS:
003F 2A0500
                             LHLD
                                       DPBMAB
                                                           ;Get maximum allocation block
0042 C32800
                                                           Return from GTAS with parameters in HL and DE
                             IMP
                                       GTAS
                   ;GDTAS
                   ;Get directory track and sector
                   ;This returns the START track and sector for the
                   ;file directory, along with the number of sectors
                   : in the directory.
                   ;>>>>> Note: You must call GETDPB before
;>>>>> you call this subroutine
                   ;Entry parameters: none
                   ;Exit parameters:
                             BC = number of sectors in directory
DE = directory start sector
HL = directory start track
                   ;
                   2
                   ;
                   GDTAS:
0045 2A0700
                             LHLD
                                       DPBNOD
                                                           ;Get number of directory entries - 1
                                                           ;Make true number of entries
;Each entry is 32 bytes long, so to
;convert to 128 byte sectors, divide by 4
0048 23
                             INX
                                       н
                                                           ;/ 2 (by shifting HL right one bit)
;/ 4
0049 CDD000
004C CDD000
004F E5
                             CALL
                                       SHLR
                             CALL
                                       SHLR
                             PUSH
                                       н
                                                           ;Save number of sectors
0050 210000
0053 CD2800
0056 C1
                                       н.о
                             LXI
                                                           ;Directory starts in allocation block 0
;HL = track, DE = sector
;Recover number of sectors
                                       GTAS
                             CALL
                             POP
                                       B
0057 C9
                             RET
```



```
: GNTAS
                 ;Get NEXT track and sector
                 :This subroutine updates the input track and sector
                 ;by one, incrementing the track and resetting the
                 sector number as required.
                 ;>>>> Note: You must call GETDPB before
                                  you call this subroutine
                 ·>>>>>
                 ; Note: you must check for end of disk by comparing
                           the track number returned by this subroutine
to that returned by by GMTAS + 1. When
                 :
                 2
                           equality occurs, the end of disk has been reached.
                 .
                 ;Entry parameters
                           HL = current track number
                 - 2
                           DE = current sector number
                 .
                 ;Exit parameters
                           HL = updated track number
DE = updated sector number
                 .
                 GNTAS:
0058 E5
                           PUSH
                                     н
                                                        :Save track
0059 13
                           INX
                                     D
                                                        ;Update sector
005A 2A0000
                           LHLD
                                     DPBSPT
                                                        ;Get sectors per track
005D CDC900
                           CALL
                                     SUBHL
                                                        ;HL = HL - DE
0060 E1
                           POP
                                     н
                                                        ;Recover current track
                           RNC
                                                        ;Return if updated sector <= sec. per trk.
;Update track if upd. sec > sec. per trk.
;Reset sector to 1
0061 DO
0062 23
0063 110100
                           TNY
                                     н
                                     D, 1
                           LXI
0066 C9
                           RET
                  ;GAB
                  ;Get allocation block
                  ;This subroutine returns an allocation block number
                  ; given a specific track and sector. It also returns ; the offset down the allocation block at which the
                  ;sector will be found. This offset is in units of
                  ;128-byte sectors.
                  :>>>>>
                            Note: You must call GETDPB before
                  ;>>>>>
                                    you call this subroutine
                  ;Entry parameters
                           HL = track number
DE = sector number
                  :
                  ;Exit parameters
                           HL = allocation block number
                  :
                  :Method
                  ;The allocation block is formed from:
;AB = (sector + ((track - tracks before directory)
                           * sectors per track)) / log2 (sectors per all. block)
                  ;The sector offset within allocation block is formed from:
                  GAB:
0067 D5
0068 EB
                                                        ;Save sector
;DE = track
                           PUSH
                                     D
                           XCHG
                                                        ;Get no. of tracks before directory
0069 2A0D00
006C EB
                                     DPBTBD
                           IHID
                                                        ;DE = no. of tracks before directory
;DE = no. of tracks before dir. HL = track
;HL = HL - DE
                            XCHG
006D CDC900
                           CALL
                                     SUBHL
                                                        ;HL = relative track within logical disk
                                                        ;DE = relative track
0070 EB
                           XCHG
0071 2A0000
0074 CDA400
                           LHLD
                                     DPRSPT
                                                        ;Get sectors per track
                           CALL
                                     MULHL
                                                        ;HL = HL * DE
                                                        ;HL = number of sectors
;DE = number of sectors
0077 FB
                           XCHG
```

Figure 5-24. (Continued)

	0078	3 E1		POP	н	;Recover sector
	0075	2B		DCX	н	Make relative to 0
	007A	19		DAD	D	;HL = relative sector
		340300		LDA	DPBBM	;Get block mask
	0075	47		MOV	B, A	;Ready for AND operation
	007E 007F	70		MOV	A,L	;Get LS byte of relative sector
	0080	10				AND with block work
				ANA	B	;AND with block mask
	0081			PUSH	PSW	;A = sector displacement
		340200		LDA	DPBBS	;Get block shift
	0085	4F		MOV	C,A	;Make into counter
			GABS:			;Shift loop
	0086	CDD000		CALL	SHLR	;HL shifted right (divided by 2)
	0089	OD		DCR	с	;Count down
		C28600		JNZ	GABS	;Shift again if necessary
	0080	F1		POP	PSW	Recover offset
	008E			RET		filecover office
		•••				
				y subrou	tinge	
						a an Aba IN secondation and s
			; mese	periorm	10-Dit arithmeti	c on the HL register pair.
			;DIVHL			
					DE using an iter-	
			;In pra	ctice, i	t uses an iterat	ive ADD of the complemented divisor.
			;Entry	paramete	rs	
			;	HL = di	vidend	
			;	DE = di	visor	
			:Exit p	arameter	5	
				BC = qu		
				HL = re		
			,		arnaer	
			DIVHL:			
	008F	DS.	DIVIC.	PUSH	D	- Cause addresses
	000F	55		FUSH	D	Save divisor
						;Note : 2's complement is formed by
						; inverting all bits and adding 1.
	0090			MOV	A,E	;Complement divisor (for iterative
	0091	2F		CMA		;ADD later on)
	0092			-MOV	E,A	
	0093	7A		MOV	A,D	;Get MS byte
	0094	2F		CMA		;Complement it
	0094 0095	57		MOV	D,A	
	0096			INX	D	;Make 2's complement
						;Now, subtract negative divisor until
						dividend goes negative, counting the number
						;of times the subtract occurs
	0097	010000		LXI	В,О	;Initialize quotient
	••••		DIVHLS:		270	;Subtract loop
	009A	03		INX	в	;Add 1 to quotient
	009B			DAD	D	;"Subtract" divisor
		DA9A00		JC	DIVHLS	
	5070	DHZMUU		00	DIVNES	;Dividend not yet negative
	0005	0.D		DOV	р	;Dividend now negative, quotient 1 too large
	009F	OB		DCX	В	;Correct quotient
		CD		VOUC		;Compute correct remainder
	0040			XCHG		;DE = remainder - divisor
	00A1			POP	Н	Recover positive divisor
	00A2			DAD	D	;HL = remainder
	00A3	C9		RET		;BC = quotient, HL = remainder
			; MULHL			
			;Multipl	ly HL × I	DE using iterativ	/e ADD.
			;Entry F	arameter	~ 5	
					ltiplicand	
			,	DE = mul		
			•			
			— · · ·			
				rameters		
			;	HL = pro		
			;	DE ≈ mul	ltiplier	
			MULHL:			
	00A4	C5		PUSH	В	;Save user register
						;Check if either multiplicand
						; or multiplier is O
_						

Figure 5-24. (Continued)

0045	70		MOU	х Ц	
00A5 00A6	7C. B5		MOV ORA	A,H L	
00A7	CAC400		JZ	MULHLZ	;Yes, fake product
00AA 00AB			MOV ORA	A,D F	
	CAC400		JZ	MULHLZ	;Yes, fake product
					;This routine will be faster if ; the smaller value is in DE
00AF 00B0			MOV CMP	A, D H	;Get MS byte of current DE value ;Check which is smaller
	DAB500		JC	MULHLN	;C set if D < H, so no exchange
00B4	EB		XCHG		
	•••	MULHLN:	MOUL		- PC
00B5 00B6			MOV MOV	B,D C,E	;BC = multiplier
00B7 00B8			MOV MOV	D,H	;DE = HL = multiplicand
00B8	OB		DCX	E,L B	;Adjust count as
					;1 * multiplicand = multiplicand
		MULHLA:			;ADD loop
OOBA			MOV	А, В С	;Check if all iterations completed
00BB 00BC	CAC700		ORA JZ	MULHLX	;Yes, exit
OOBF	19		DAD	D	;HL ≈ multiplicand + multiplicand
0000			DCX	B	Countdown on multiplier - 1
0001	C3BA00		JMP	MULHLA	;Loop back until all ADDs done
		MULHLZ:			
00C4	210000		LXI	н,о	;Fake product as either multiplicand ; or multiplier is O
		MULHLX:			
0007	C1	MULHLX:	POP	в	;Recover user register
0008			RET		
		;SUBHL ;Subtra	ctHL – 1	DE	
		:Entry (parameter	r 5	
		;	HL = sul	otrahend otractor	
		;Exit p ;	arameter HL = di	s fference	
		SUBHL:			
00C9 00CA	7D		MOV	A,L	;Get LS byte
00CA 00CB	93		SUB MOV	E L,A	;Subtract without regard to carry ;Put back into difference
0000	6F 7C		MOV	A, H	;Get MS byte
OOCD	9A		SBB	D	;Subtract including carry
00CE 00CF			MOV RET	H, A	;Move back into difference
000.	0,	;SHLR			
			HL right	one place (divi	ding HL by 2)
		;Entry ;	paramete HL = va	rs lue to be shifte	d È
		;Exit p ;	arameter HL = va		
		SHLR:			
00D0 00D1	B7		ORA	A	;Clear carry
00D1 00D2			MOV RAR	А,Н	;Get MS byte ;Bit 7 set from previous carry,
0002	1				; bit O goes into carry
00D3			MOV	Н, А	;Put shift MS byte back
0000			MOV RAR	A,L	;Get LS byte ;Bit 7 = bit 0 of MS byte
00D5 00D6			MOV	L,A	;Bit / = bit 0 of MS byte ;Put back into result
0007			RET		

Figure 5-24. (Continued)

Function 32: Set/Get User Number

Function Code:	C = 20H
Entry Parameters:	E = 0FFH to get user number, or
	E = 0 to 15 to set user number
Exit Parameters:	A = Current user number if E was 0FFH

Example

0020 =	B\$SETGETUN	EQU 32	;Set/Get User Number
0005 =	BDOS	EQU 5	;BDOS entry point
0000 0E20 0002 1E0F 0004 CD0500 0007 0E20 0009 1EFF 0008 CD0500	MVI MVI CALL MVI CALL	C, B\$SETGETUN E, 15 BDOS C, B\$SETGETUN E, OFFH BDOS	;To set user number ;Function code ;Required user number ;To get user number ;Function code ;Indicate request to GET ;A = Current user no. (0 15)

Purpose This subroutine either sets or gets the current user number. The current user number determines which file directory entries are matched during all disk file operations.

When you call this function, the contents of the E register specify what action is to be taken. If E = 0FFH, then the function will return the current user number in the A register. If you set E to a number in the range 0 to 15 (that is, a valid user number), the function will set the current user number to this value.

Notes You can use this function to share files with other users. You can locate a file by attempting to open a file and switching through all of the user numbers. Or you can share a file in another user number by setting to that number, operating on the file, and then reverting back to the original user number.

If you do change the current user number, make provisions in your program to return to the original number before your program terminates. It is disconcerting for computer operators to find that they are in a different user number after a program. Files can easily be damaged or accidentally erased this way.

Function 33: Read Random

Function Code: C = 21 HEntry Parameters: DE = Address of FCBExit Parameters: A = Return code

Example

0021 =	B\$READRAN	EQU	33 ;Read Random
0005 =	BDOS	EQU	5 ;BDOS entry point
	50D.		File control block
	FCB:		FILE CONTROL DIOCK
0000 00	FCB\$DISK:	DB	0 ;Search on default disk drive
0001 464940454	4EFCB\$NAME:	DB	'FILENAME' ;File name
0009 545950	FCB\$TYP:	DB	TYP' ;File type

000F 0010 0018 0020	0000 00 0000000000 000000000 00 00 0000	FCB\$EXTENT: FCB\$RESV: FCB\$RECUSED: PCB\$ABUSED: FCB\$EQREC: FCB\$RANREC: FCB\$RANREC:	DB DB DB DB DB DB DB DW DB		;Extent ;Reserved for CP/M ;Records used in this extent ,0,0,0,0 ;Allocation blocks used ,0,0,0,0 ;Sequential rec. to read/write ;Random rec. to read/write ;Random rec. overflow byte (MS)
0024	D204	RANRECNO:	DW	1234	;Example random record number
0029 002C 002E	2A2400 222100 0E21 110000 CD0500	LHLD SHLD MVI LXI CALL	RANRECN FCB\$RAN C,B\$REA D,FCB BDOS	REC	<pre>;Record will be read into ; address set by prior ; SETDMA call ;Get random record number ;Set up file control block ;Function code ;DE -> file control block ;A = 00 if operation successful ;A = nonzero if no data in ; file specifically: ;A = 01 attempt to read ; 03 CP/M could not ; close current extent ; 04 attempt to read ; unwritten extent ; 06 attempt to read ; beyond end of disk</pre>

Purpose This function reads a specific CP/M record (128 bytes) from a random file that is, a file in which records can be accessed directly. It assumes that you have already opened the file, set the DMA address using the BDOS Set DMA function, and set the specific record to be read into the random record number in the FCB. This function computes the extent of the specified record number and attempts to open it and read the correct CP/M record into the DMA address.

The random record number in the FCB is three bytes long (at relative bytes 33, 34, and 35). Byte 33 is the least significant byte, 34 is the middle byte, and 35 the most significant. CP/M uses only the most significant byte (35) for computing the overall file size (function 35). You must set this byte to 0 when setting up the FCB. Bytes 33 and 34 are used together for the Read Random, so you can access from record 0 to 65535 (a maximum file size of 8,388,480 bytes).

This function returns with A set to 0 to indicate that the operation has been completed successfully, or A set to a nonzero value if an error has occurred. The error codes are as follows:

- A = 01 (attempt to read unwritten record)
- A = 03 (CP/M could not close current extent)
- A = 04 (attempt to read unwritten extent)
- A = 06 (attempt to read beyond end of disk)

Unlike the Read Sequential BDOS function (code 20, 14H), which updates the current (sequential) record number in the FCB, the Read Random function leaves the record number unchanged, so that a subsequent Write Random will replace the record just read.

You can follow a Read Random with a Write Sequential (code 21, 15H). This

will rewrite the record just read, but will then update the sequential record number. Or you may choose to use a Read Sequential after the Read Random. In this case, the same record will be reread and the sequential record number will be incremented. In short, the file can be sequentially read or written once the Read Random has been used to position to the required place in the file.

Notes

To use the Read Random function, you must first open the *base extent* of the file, that is, extent 0. Even though there may be no actual data records in this extent, opening permits the file to be processed correctly.

One problem that is not immediately obvious with random files is that they can easily be created with gaps in the file. If you were to create the file with record number 0 and record number 5000, there would be no intervening file extents. Should you attempt to read or copy the file sequentially, even using CP/M's file copy utility, only the first extent (and in this case, record 0) would get copied. A Read Sequential function would return an "end of file" error after reading record 0. You must therefore be conscious of the type of the file that you try and read.

See Figure 5-26 for an example subroutine that performs Random File Reads and Writes. It reads or writes records of sizes other than 128 bytes, where necessary reading or writing several CP/M records, prereading them into its own buffer when the record being written occupies only part of a CP/M record. It also contains subroutines to produce a 32-bit product from multiplying HL by DE (MLDL—Multiply double length) and a right bit shift for DE, HL (SDLR—Shift double length right).

Function 34: Write Random

Function Code:C = 22HEntry Parameters:DE = Address of file control blockExit Parameters:A = Return code

Example

0022 =	B\$WRITERAN	EQU	34	Write Random
0005 =	BDOS	EQU	5	;BDOS entry point
	FCB:			;File control block
0000 00	FCB\$DISK:	DB	0	;Search on default disk drive
0001 46494045	54EFCB\$NAME:	DB	'FILEN	AME ;File name
0009 545950	FCB\$TYP:	DB	TYP	;File type
00 2000	FCB\$EXTENT:	DB	0	:Extent
000D 0000	FCB\$RESV:	DB	0.0	Reserved for CP/M
000F 00	FCB\$RECUSED:	DB	0	Records used in this extent
0010 00000000	OOFCB\$ABUSED:	DB	0.0.0.	0,0,0,0,0 ;Allocation blocks used
0018 00000000	000	DB		0.0.0.0.0
0020 00	FCB\$SEQREC:	DB	0	;Sequential rec. to read/write
0021 0000	FCB\$RANREC:	DW	ò	Random rec. to read/write
0023 00	FCB\$RANRECO:	DB	Ō	;Random rec. overflow byte (MS)
0024 D204	RANRECNO:	DW	1234	;Example random record number
				Record will be written from
				; address set by prior

; SETDMA call

0026 2A2400 0029 222100 002C 0E22 002E 110000 0031 CD0500	LHLD SHLD MVI LXI CALL	RANRECNO FCB\$RANREC C,B\$₩RITERAN D,FCB BDOS	;Get random record number ;Set up file control block ;Function code ;DE -> file control block ;A = 00 if operation successful ;A = nonzero if no data in file ; specifically: ;A = 03 CP/M could not ; close current extent ; 05 directory full ; 06 attempt to write ; beyond end of disk
---	------------------------------------	---	---

Purpose This function writes a specific CP/M record (128 bytes) into a random file. It is initiated in much the same way as the companion function, Read Random (code 33, 21 H). It assumes that you have already opened the file, set the DMA address to the address in memory containing the record to be written to disk, and set the random record number in the FCB to the specified record being written. This function also computes the extent in which the specified record number lies and opens the extent (creating it if it does not already exist). The error codes returned in A by this call are the same as those for Read Random, with the addition of error code 05, which indicates a full directory.

Like the Read Random (but unlike the Write Sequential), this function does not update the logical extent and sequential (current) record number in the FCB. Therefore, any subsequent sequential operation will access the record just written by the Read Random call, but these functions will update the sequential record number. The Write Random can therefore be used to position to the required place in the file, which can then be accessed sequentially.

Notes

In order to use the Write Random, you must first open the base extent (extent 0) of the file. Even though there may be no data records in this extent, opening permits the file to be processed correctly.

As explained in the notes for the Read Random function, you can easily create a random file with gaps in it. If you were to create a file with record number 0 and record number 5000, there would be no intervening file extents.

Figure 5-25 shows an example subroutine that creates a random file (CRF) but avoids this problem. You specify the number of 128-byte CP/M records in the file. The subroutine creates the file and then writes zero-filled records throughout. This makes it easier to process the file and permits standard CP/M utility programs to copy the file because there is a data record in every logical record position in the file. It is no longer a "sparse" file.

Figure 5-26 shows a subroutine that ties the Read and Write Random functions together. It performs Random Operations (RO). Unlike the standard BDOS functions that operate on 128-byte CP/M records, RO can handle arbitrary record size from one to several thousand bytes. You specify the relative record number of your record, not the CP/M record number (RO computes this). RO also prereads a CP/M record when your logical record occupies part of a 128-byte record, either because your record is less than 128 bytes or because it spans more than one

```
; CRF
                ;Create random file
                ;This subroutine creates a random file. It erases any previous
                ;file before creating the new one, and then writes O-filled
                precords throughout the entire file.
               ;Entry parameters
; DE -> file control block for new file
                :
                        HL = Number of 128-byte CP/M records to be
                                zero-filled.
                :
                ;Exit parameters
                        Carry clear if operation successful (A = 0, 1, 2, 3)
Carry set if error (A = OFFH)
                ;Calling sequence
; LXI D.FCB
                        CALL
                ;
                                CRF
                                ERROR
                :
                        JC:
0013 =
                B$ERASE
                                EQU
                                         19
                                                 :Erase file
0016 =
                B$CREATE
                                EQU
                                         22
                                                 ;Create file
001A =
                B$SETDMA
                                EQU
                                         26
                                                 ;Set DMA address
0015 =
                B$WRITESEQ
                                EQU
                                         21
                                                 ;Write sequential record
0005 =
                BDOS
                                EQU
                                         5
                                                 ;BDOS entry point
                CRFBUF:
                                                 ;Zero-filled buffer
0000 000000000
                        nω
                                0,0,0
0032 0000000000
                        DW
                                0.0.0
0064 0000000000
                                0,0,0,0,0,0,0,0,0,0,0,0,0,0
                        DW
0080 0000
               CRFRC:
                        DW
                                0
                                                 :Record count
                CRF:
0082 228000
                        SHLD
                                CRFRC
                                                 ;Save record count
0085 D5
                        PUSH
                                D
                                                 Preserve FCB pointer
0086 0E13
                        MVI
                                C, B$ERASE
                                                 ;Erase any existing file
0088 CD0500
                        CALL
                                BDOS
008B D1
                        POP
                                n
                                                 ;Recover FCB pointer
008C D5
                        PUSH
                                D
                                                 ; and resave
008D 0E16
008F CD0500
                        MUT
                                C.B$CREATE
                                                 ;Create (and open new file)
                        CALL
                                BDOS
0092 FEFF
                        CPI
                                                 ;Carry set if ÖK, clear if error
                                OFFH
0094 3F
                                                 ;Complete to use carry set if error
;Recover FCB address
                        CMC
0095 D1
                        POP
                                D
0096 D8
                        RC
                                                 ;Return if error
0097 D5
                        PUSH 1
                                                 Resave FCB pointer
                                D
0098 0E1A
                        MVI
                                C, B$SETDMA
                                                 ;Set DMA address to O-buffer
009A 110000
009D CD0500
                        LXI
                                D, CRFBUF
                        CALL
                                BDOS
00A0 D1
                        POP
                                n
                                                 ;Recover FCB pointer
               CRFL:
00A1 2A8000
                                CRFRC
                        LHLD
                                                 ;Get record count
00A4 7D
                        MOV
                                A.L
00A5 B4
                        ORA
                                н
                                                 :Check if count now zero
00A6 C8
                        RZ
                                                 ;Yes, exit
00A7 2B
                        DCX
                                н
                                                 ;Downdate count
                                CRFRC
00A8 228000
                        SHLD
                                                 ;Save count
OOAB D5
                        PUSH
                                D
                                                 Resave FCB address
00AC 0E15
                        MVI
                                C, B$WRITESEQ
                                                 ;Write sequentially
00AE CD0500
                        CALL
                                BDOS
                                                 ;Recover FCB
00B1 D1
                        POP
                                n
                                CRFL
00B2 C3A100
                        JMP
                                                 ;Write next record
```



128-byte sector. The subroutine suppresses this preread if you happen to use a record size that is some multiple of 128 bytes. In this case, your records will fit exactly onto a 128-byte record, so there will never be some partially occupied 128-byte sector.

This example also contains subroutines to produce a 32-bit product from multiplying HL by DE (MLDL—Multiply double length) and a right bit shift for DE, HL (SDLR—Shift double length right).

	;RO :Random operation (read or write)					
	Manuoli operation (read of write)					
	This subroutine reads or writes a random record from a file.					
	The record length can be other than 128-bytes. This					
	;subroutine computes the start CP/M record (which					
				g, performs a random read		
				record into a user buffer.		
	; user-specifie			s will be read until the complete		
				ne user-specified record is not an exact		
				appropriate sectors will be preread.		
	;It is not nec	essary to	preread	d when the user-specified record		
				when subroutine is processing		
	;CP/M records	entirely	spanned	by a user-specified record.		
	· Fature assessed					
	;Entry paramet		block (of the form:		
	; nu - / ;	DB	0	;OFFH when reading, OOH for write		
		DW	FCB	Pointer to FCB		
		DW	RECNO	User record number		
	;	DW	RECSZ	User record size		
	;	DW	BUFFER			
	;			; RECSZ bytes in length		
	;Exit paramete					
			ion com	pleted (and user record		
	:			er buffer)		
	; 1	if attemp	t to rea	ad unwritten CP/M record		
	; 3	if CP/M c	t close an extent			
				ad unwritten extent		
	; 5 if CP/M could not					
	; 6	it attemp	t to rea	ad beyond end of disk		
	Calling seque	nce				
	; LXI	H, PARAM	IS	;HL -> parameter block		
	; CALL	RO				
	; ORA	A		;Check if error		
	; JNZ	ERROR				
0021 =	FCBE\$RANREC	EQU	33	;Offset of random record no. in FCB		
001A =	B\$SETDMA	EQU	26	;Set the DMA address		
0021 =	B\$READRAN	EQU	33	;Read random record		
0028 =	B\$WRITERANZ	EQU	40	;Write random record with zero-fill		
1				; previously unallocated allocation		
			_	; blocks		
0005 =	BDOS	EQU	5	BDOS entry point		
	ROPB:			;Parameter block image		
0000 00	ROREAD: DB	0		;NZ when reading, Z when writing		
0001 0000	ROFCB: DW	0		Pointer to FCB		
0003 0000	ROURN: DW	0		;User record number		
0005 0000	ROURL: DW	0		User record length		
0007 0000	ROUB: DW	0		;Pointer to user buffer		
0009 =	ROPBL EQU	\$-ROPB		;Parameter block length		
0009 0000	ROFRP: DW	0		Pointer to start of user record fragment		
		÷		; in first CP/M-record read in		

Figure 5-26. Read/Write variable length records randomly

000B 00 000C 00 000E 00	000	ROFRL: RORNP: ROWECR:	DB DW DB	0 0 0	;Fragment length ;Record number pointer (in user FCB) ;NZ when writing user records that are an ; exact super-multiple of CP/M-record (and ; therefore no preread is required)
000F		ROBUF:	DS	128	;Buffer for CP/M record
008F 1 0092 08 0094 CI		RO:	LXI MVI CALL	D,ROPB C,ROPBL MOVE	;DE -> local parameter block ;Parameter block length ;Move C bytes from HL to DE
			; compu ; of the ; user ; ; signi ; byte ; ;The pro	te the relative e user record wi record number *i ficant 7 bits of offset of the st oduct / 128 (shi	ser record in CP/M record, BYTE offset of the start thin the file (i.e. record size). The least this product give the art of the user record. fted left 7 bits) gives the the start of the user record.
0097 24	40500		LHLD	ROURL	;Get user record length
009A 7I			MOV	A,L	;Get LS bytes of user rec. length
009B E6			ANI ORA	7FH A	;Check if exact multiple of 128 ;(i.e. exact CP/M records)
009E 38	E00		MVI	A, 0	;A = 0, flags unchanged
00A0 C2 00A3 30			JNZ DCR	RONE	;Not exact CP/M records ;A =FF
00A4 32	20E00	RONE :	STA	ROWECR	;Set write-exact-CP/M-records flag
00A7 EE	3		XCHG		;DE = user record length
00A8 24 00AB CE	40300		LHLD	ROURN	;Get user record number
UUAB CL	19801		CALL	MLDL	;DE,HL = HL * DE ;DE,HL = user-record byte offset in file
OOAE DS			PUSH	D	;Save user-record byte offset
OOAF ES			PUSH	н	
00B0 7E			MOV ANI	A,L 7FH	;Get LS byte of product ;Isolate byte offset within
00B3 4F			MOV MVI	C,A B,O	;CP/M record ;Make into word value
00B6 21			LXI	H, ROBUF	;Get base address of local buffer
00B9 09			DAD	B	;HL -> Start of fragment in buffer
00BA 22	20900		SHLD	ROFRP	;Save fragment pointer
			;remaind		nt length that could reside in rd, based on the offset in the fragment starts.
00BD 47			MOV	B,A	;Take copy of offset in CP/M record
00BE 3E			MVI SUB	A,128 B	;CP/M record size
0000 90			STA	ROFRL	;Compute 128 - offset ;Assume this is the fragment length
					gth is less than the assumed it in place of the result above
00C4 47	,		MOV	B.A	;Get copy of assume frag. length
00C5 3A	0600		LDA	ROURL+1	;Get MS byte of user record length
00C8 B7			ORA	A	;If NZ, rec. len. must be > 128 ;So fragment length is OK
00C9 C2			JNZ LDA	ROFLOK ROURL	;So fragment length is UK ;Still a chance that rec. len.
OOCF BS	1		CMP	в	; less than fragment len.
OODO D2	20600			ROFLOK	;NC if user rec. len. => frag. len.
00D3 32	0800		STA	ROFRL	;User rec. len. < frag. len. so ; reset fragment length to smaller
		ROFLOK:			
00D6 3A		NUPLOK:	LDA	ROWECR	;Get exact CP/M record flag
00D9 47	,		MOV	B,A	;for ANDing with READ flag
OODA 3A			LDA	ROREAD	;Get read operation flag
OODD 2F			CMA		;Invert so NZ when writing

Figure 5-26. (Continued)

00DE A0 00DF 320E00		ANA STA	B ROWECR	;Form logical AND ;Save back in flag
		; of the	e start of the us	ngth byte offset within the file ser record. Shift 7 places right get the CP/M record number for record.
00E2 E1 00E3 D1		POP POP	H D	Recover user rec. byte offset
00E3 D1 00E4 0E07		MVI	Б С,7	;Count for shift right
00E6 CDF101	ROS:	CALL	SDLR	:DE.HL = DE.HL / 2
00E9 0D 00EA C2E600			C ROS	
OOED 7A		MOV	A, D	;Error if DE still NZ after
OOEE B3 OOEF C2AC01		ORA JNZ	E ROERO	; division by 128.
00F2 EB 00F3 2A0100 00F6 012100 00F9 09 00FA 220C00 00FD 73		XCHG LHLD LXI DAD SHLD MOV	ROFCB B,FCBE\$RANREC B RORNP M,E	;Set CP/M record number in FCB ;DE = CP/M record number ;Get pointer to FCB ;Offset of random record no. in FCB ;HL -> ran. rec. no. in FCB ;Save record number pointer ;Store LS byte
00FE 23 00FF 72		INX MOV	H M,D	;Store MS byte
0100 OE1A 0102 110F00 0105 CD0500		MVI LXI CALL	C,B\$SETDMA D,ROBUF BDOS	;Set DMA address to local buffer
0108 3A0E00 0108 87 010C C21F01		LDA ORA JNZ	'ROWECR A ROMNF	;Bypass preread if exact sector write
010F 2A0100 0112 EB		LHLD XCHG	ROFCB	;Get pointer to FCB ;DE -> FCB
0113 0E21 0115 CD0500		MVI CALL	C,B\$READRAN BDOS	;Read random function
0118 FE05 011A DCAF01		CPI CC	5 ROCIE	;Check if error code < 5 ;Yes, check if ignorable error ; (i.e. error reading unwritten part ; of file for write operation preread)
011D B7 011E CO		ORA RNZ	A	;Check if error ;Yes
011F 2A0700 0122 EB 0123 2A0900 0126 3A0B00 0129 4F	ROMNF:	LHLD XCHG LHLD LDA MOV	ROUB ROFRP ROFRL C,A	;Move next fragment ;Get pointer to user buffer ;DE -> user buffer ;HL -> start of user rec. in local buffer ;Get fragment length ;Ready for MOVE
012A 3A0000 012D B7		LDA ORA	ROREAD A	;Check if reading
012E C23201 0131 EB		JNZ XCHG	RORD 1	;Yes, so leave DE, HL unchanged ;Writing, so swap source and destination ;DE -> start of user rec. in local buffer ;HL -> user buffer
0132 CDFE01	RORD1:	CALL	MOVE	;Reading - fragment local -> user buffer
0132 CDFE01		LDA	ROREAD	;Wrading - fragment local -> user buffer ;Writing - fragment user -> local buffer ;Check if writing
0138 B7 0139 CA3D01 013C EB		ORA JZ XCHG	A ROWR1	;Writing, so leave HL -> user buffer ;HL -> next byte in user buffer
013D 220700 0140 3A0000	ROWR1:	SHLD LDA	ROUB ROREAD	;Save updated user buffer pointer ;Check if reading

Figure 5-26. (Continued)

0143 B7 0144 C25001		ORA JNZ	A RORD3	;Yes, bypass write code
0147 0E28		MVI	C,B\$WRITERANZ	;Write random
0149 2A0100		LHLD	ROFCB	;Get address of FCB
014C EB		XCHG		;DE -> FCB
014D CD0500		CALL	BDOS	
	RORD3:	;If ne ;more ;the s ;lengt ;the n	cessary (because CP/M records wil tart of the frag h depends on whe ext sector or sp	th of user record as yet unmoved. more data needs to be transferred) l be read. In this case ment will be offset 0. The fragment ther the user record finishes within ans it. If the residual length of the the fragment length will be set to
0150 2A0500		LHLD	ROURL	;Get residual user rec. length
0153 3A0B00		LDA	ROFRL	;Get fragment length just moved
0156 5F		MOV	E,A	;Make into a word value
0157 1600		MVI	D, O	
0159 CDEA01		CALL	SUBHL	;Compute ROURL - ROFRL
015C 7C		MOV	А, Н	;Check if result O
015D B5		ORA	L	
015E C8		RZ		;Return when complete USER
				; record has been transferred
015F 220500		SHLD	ROURL	Save downdated residual rec. length
0162 4D		MOV	C,L	Assume residual length < 128
0163 118000 0166 CDEA01		LXI CALL	D,128 SUBHL	;Check if residual length is < 128 ;HL = HL - DE
0169 FA6E01		JM	ROLT128	;negative if < 128
016C 0E80		MVI	C,128	;=> 128, so set frag.length to 128
	DO: 24-5	•		
D16E 79	ROLT128	B: MOV	A,C	
016F 320B00		STA	ROFRL	;Fragment length now is either 128
0172 210F00		LXI	H, ROBUF	<pre>; if more than 128 bytes left to input ; in user record, or just the right ; number of bytes (< 128) to complete ; the user record. ;All subsequent CP/M records will start</pre>
0175 220900		SHLD	ROFRP	; at beginning of buffer
				;Update random record number in FCB
0178 2A0C00		LHLD	RORNP	;HL -> random record number in user FCB
017B 5E		MOV	E, M	;Increment the random record number
017C 23		INX	H	;HL -> MS byte of record number ;Get MS byte
017D 56 017E 13		INX	D, M D	;Get MS byte ;Update record number itself
017F 7A		MOV	A.D	Check if record now 0
0180 B3		ORA	E E	, then in receive now v
0181 C28701		JNZ	ROSRN	;No, so save record number
0184 3E06		MVI	A,6	;Indicate "seek past end of disk"
D186 C9		RET		Return to user
	ROSRN:			· · ·
0187 72		MOV	M, D	;Save record number
0188 2B		DCX	H	;HL -> LS byte
0189 73		MOV	M,E	
				;If writing, check if preread required
018A 3A0E00		LDA	ROWECR	;Check if exact CP/M record write
D18D B7		ORA	A	
018E C21F01		JNZ	ROMNF	;Yes, go move next fragment
0191 3A0000		LDA	ROREAD	;If reading, perform read unconditionally
019 4 B7		ORA	Α	
0195 C2A001		JNZ	RORD2	
0198 3A0B00		LDA	ROFRL	;For writes, bypass preread if
		CPI	128	; whole CP/M-record is to be overwritten
019B FE80		JZ	ROMNE	; (fragment length = 128)
	DODDC			
019D CAIFOI	RORD2:	MUT		:Read the pext CP/M record
019B FE80 019D CA1F01 01A0 0E21 01A2 2A0100	RORD2:	MVI LHLD	C,B\$READRAN ROFCB	;Read the next CP/M record ; in sequence

Figure 5-26. (Continued)

01A5 EB 01A6 CD0500		XCHG CALL	BDOS	;DE -> FCB
01A9 C31F01		JMP	ROMNF	;Go back to move next fragment
	ROERO:			;Error because user record number ; * User record length / 128 gives ; a CP/M record number > 65535.
01AC 3E04		MVI	A, 4	;Indicate "attempt to read unwritten
01AE C9		RET		; extent"
	ROCIE:			;Check ignorable error (preread ; for write operation)
01AF 47 01B0 3A0000		MOV LDA	B,A ROREAD	;Save original error code ;Check if read operation
01B3 B7 01B4 78		ORA MOV	А А, В	Restore original error code but
01B5 C0		RN7		; leave flags unchanged ;Return if reading
0186 AF		XRA	Α	;Fake "no error" indicator
01B7 C9		RET		
	;MLDL ;Multip ;return	ly HL ∗ ≥d in DE	DE using iterat ,HL.	ive ADD with product
	;Entry ; ;	HL = mu	rs Itiplicand Itiplier	
	;Exit p ; ;	DE,HL =	s Product Itiplier	
	MLDL:			
01B8 010000 01BB C5		LXI PUSH	В, О В	;Put 0 on top of stack ; to act as MS byte of product ;Check if either multiplicand ; or multiplier is 0
01BC 7C		MOV	A,H	; of multiplier is o
01BD B5 01BE CAE501		ORA JZ	L MLDLZ	;Yes, fake product
01C1 7A		MOV	A,D	
01C2 B3 01C3 CAE501		ORA JZ	E MLDLZ	;Yes, fake product
				;This routine will be faster if
				; the smaller value is in DE
01C6 7A 01C7 BC		MOV CMP	A, D H	;Get MS byte of current DE value ;Check which is smaller
01C8 DACCO1		JC	MLDLNX	;C set if D < H, so no exchange
O1CB EB	MLDLNX:	XCHG		
01CC 42 01CD 4B		MOV MOV	B,D C,E	;BC = multiplier
01CE 54 01CF 5D		MOV MOV	D,H E,L	;DE = HL = multiplicand
OIDO OB		DCX	В	;Adjust count as ; 1 * multiplicand = multiplicand
0101 70	MLDLA:	MOV	A B	;ADD loop ;Check if all iterations completed
01D1 78 01D2 B1		ORA	A,B C	, oneck in all iterations completed
01D3 CAE801		JZ	MLDLX	;Yes, exit
01D6 19 01D7 E3		DAD XTHL	D	;HL = multiplicand + multiplicand ;HL = MS bytes of result, TOS = part prod.
01D8 7D		MOV	A,L	;Get LS byte of top half of product
01D9 CE00 01DB 6F		ACI MOV	0 L.A	;Add one if carry set ;Replace
01DB 6F 01DC 7C		MOV	A, H	Repeat for MS byte
		ACI MOV	0 H, A	
01DD CEOO				
			B	;Countdown on multiplier - 1

Figure 5-26. (Continued)

I

01E5 210000	MLDLZ:	LXI	н, о	;Fake product as either multiplicand ; or multiplier is O
01E8 D1 01E9 C9	MLDLX:	POP RET	D	Recover MS part of product;
	;SUBHL ;Subtra	ict HL -	DE.	
	;Entry ; ;		ers Ibtrahend Ibtractor	
	;Exit p ;	arameter HL = di	s. fference	
01EA 7D 01EB 93 01EC 6F 01ED 7C 01EE 9A 01EF 67 01FF 67	SUBHL:	MOV SUB MOV MOV SBB MOV RET	A,L E L,A A,H D H,A	;Get LS byte ;Subtract without regard to carry ;Put back into difference ;Get MS byte ;Subtract including carry ;Move back into difference
	;SDLR ;Shift	DE,HL ri	ight one place (c	lividing DE,HL by 2)
	;Entry ;	paramete DE,HL =	ers = value to be shi	fted
	;Exit p ;	DE,HL'	-s = value / 2	
	SDLR:			
01F1 B7 01F2 EB		ORA XCHG	A	;Clear carry ;Shift DE first
01F3 CDF701 01F6 EB		CALL XCHG	SDLR2	;Now shift HL
				;Drop into SDLR2 with carry ; set correctly from LS bit
	SDLR2:			; of DE ;Shift HL right one place
01F7 7C 01F8 1F		MOV RAR	А, Н	;Get MS byte ;Bit 7 set from previous carry, ;Bit 0 goes into carry
01F9 67 01FA 7D		MOV MOV	H, A A, L	;Put shift MS byte back ;Get LS byte
01FB 1F		RAR		;Bit 7 = bit 0 of MS byte
01FC 6F 01FD C9		NOV	L,A	;Put back into result
	;MOVE ;Moves	C bytes	from HL to DE	
	MOVE:			
01FE 7E 01FF 12		MOV STAX	A, M D	;Get source byte ;Store in destination
0200 13 0201 23		INX INX	р Н	;Update destination pointer ;Update source pointer
		DCR JNZ	C MOVE	;Downdate count ;Get next byte
0202 0D 0203 C2FE01		RET		your ment byte

Figure 5-26. (Continued)

Function 35: Get File Size

Function Code:	C = 23H
Entry Parameters:	DE = Address of FCB
Exit Parameters:	Random record field set in FCB

Example

0023 0005		B\$GETFSIZ BDOS	EQU EQU	35 5	;Get Random File LOGICAL size ;BDOS entry point
		FCB:			;File control block
0000	60	FCB\$DISK:	DB	0	Search on default disk drive
	4649404546		DB	FILENAM	E' :File name
	545950	FCB\$TYP:	DB	TYP'	:File type
0000		FCBSEXTENT:	DB		Extent
0000		FCB\$RESV:	DB	-	Reserved for CP/M
000F		FCB\$RECUSED:	DB	•	Records used in this extent
		FCB\$ABUSED:	DB		0,0,0,0 ;Allocation blocks used
0018	000000000000)	DB	0,0,0,0,	
0020	00	FCB\$SEQREC:	DB	0	;Sequential rec. to read/write
0021	0000	FCB\$RANREC:	DW	0	Random rec. to read/write
0023	00	FCB\$RANRECO:	DB	Ō	;Random rec. overflow byte (MS)
0024	0E23	MVI	C.B\$GET	SIZ	Function code
0026	110000	LXI	D.FCB		:DE -> file control block
	CD0500	CALL	BDOS		,
		LHLD	FCB\$RANE		:Get random record number
0020	2A2100		F UÐ⊅RANF	λEC.	;HL = LOGICAL file size ; i.e. the record number of the ; last record

Purpose This function returns the virtual size of the specified file. It does so by setting the random record number (bytes 33-35) in the specified FCB to the maximum 128-byte record number in the file. The virtual file size is calculated from the record address of the record following the end of the file. Bytes 33 and 34 form a 16-bit value that contains the record number, with overflow indicated in byte 35. If byte 35 is 01, this means that the file has the maximum record count of 65,536.

If the function cannot find the file specified by the FCB, it returns with the random record field set to 0.

You can use this function when you want to add data to the end of an existing file. By calling this function first, the random record bytes will be set to the end of file. Subsequent Write Random calls will write out records to this preset address.

Notes Do not confuse the virtual file size with the actual file size. In a random file, if you write just a single CP/M record to record number 1000 and then call this function, it will return with the random record number field set in the FCB to 1000—even though only a single record exists in the file.

For sequential files, this function returns the number of records in the file. In this case, the virtual and actual file sizes coincide.

Function 36: Set Random Record Number

Function Code:C = 24HEntry Parameters:DE = Address of FCBExit Parameters:Random record field set in FCB

Example

0024 =	B\$SETRANREC	EQU	36 5	;Set Random Record Number
0005 =	BDOS	EQU	5	;BDOS entry point
	FCB:			;File control block
0000 00	FCB\$DISK:	DB	0	;Search on default disk drive
0001 46494C454	EFCB\$NAME:	DB	'FILEN4	AME' ;File name
0009 545950	FCB\$TYP:	DB	TYP'	;File type
0000 00	FCB\$EXTENT:	DB	0	;Extent
000D 0000	FCB\$RESV:	DB	0,0	;Reserved for CP/M
000F 00	FCB\$RECUSED:	DB	0	Records used in this extent;
0010 00000000	OFCB\$ABUSED:	DB	0,0,0,0	0,0,0,0,0 ;Allocation blocks used
0018 00000000	00	DB	0,0,0,0	0,0,0,0,0
0020 00	FCB\$SEQREC:	DB	0	Sequential rec. to read/write
0021 0000	FCB\$RANREC:	DW	0	;Random rec. to read/write
0023 00	FCB\$RANRECO:	DB	0	;Random rec. overflow byte (MS)
				: file opened and read
				; or written sequentially
0024 0E24	MVI	C.B\$SE	ETRANREC	:Function code
0026 110000	LXI	D.FCB		:DE -> file control block
0029 CD0500	CALL	BDOS		,
002C 2A2100	LHLD	FCB\$R	ANREC	:Get random record number
				;HL = random record number
				; that corresponds to the
				; sequential progress down
				; the file.

Purpose This function sets the random record number in the FCB to the correct value for the last record read or written sequentially to the file.

Notes This function provides you with a convenient way to build an index file so that you can randomly access a sequential file. Open the sequential file, and as you read each record, extract the appropriate key field from the data record. Make the BDOS Set Random Record request and create a new data record with just the key field and the random record number. Write the new data record out to the index file.

Once you have done this for each record in the file, your index file provides a convenient method, given a search key value, of finding the appropriate CP/M record in which the data lies.

You can also use this function as a means of finding out where you are currently positioned in a sequential file—either to relate a CP/M record number to the position, or simply as a place-marker to allow a repositioning to the same place later.

Function 37: Reset Logical Disk Drive

Function Code: C = 25HEntry Parameters: DE = Logical drive bit mapExit Parameters: A = 00H

Example

0025 =	B\$RESETD	EQU	37	Reset Logical Disks;
0005 =	BDOS	EQU	5	;BDOS entry point

```
;DE = Bit map of disks to be
                                               ; reset
                                               ;Bits are = 1 if disk to be
                                                 reset
                                               ;Bits 15 14 13 ...
                                                                  210
                                               ;Disk
                                                      Р
                                                         0 N ... C B A
                       LXI
                               D.0000$0000$0000$0010B ;Reset drive B:
0000 110200
0003 0E25
                       MVI
                               C, B$RESETD
                                               ;Function code
0005 CD0500
                       CALL
                               BDOS
```

Purpose This function resets individual disk drives. It is a more precise version of the Reset Disk System function (code 13,ODH), in that you can set specific logical disks rather than all of them.

The bit map in DE shows which disks are to be reset. The least significant bit of E represents disk A, and the most significant bit of D, disk P. The bits set to 1 indicate the disks to be reset.

Note that this function returns a zero value in A in order to maintain compatibility with MP/M.

Notes Use this function when only specific diskettes need to be changed. Changing a diskette without requesting CP/M to log it in will cause the BDOS to assume that an error has occurred and to set the new diskette to Read-Only status as a protective measure.

Function 40: Write Random with Zero-fill

Function Code:C = 28HEntry Parameters:DE = Address of FCBExit Parameters:A = Return Code

Example

0028 0005		B\$WRITERANZ BDOS	EQU EQU	40 5	;Write Random with Zero-Fill ;BDOS entry point
0000	00	FCB: FCB\$DISK:	DB		;File control block :Search on default disk drive
	4649404546		DB	FILENAM	
	545950	FCBSTYP:	DB	TYP'	:File type
0000		FCBSEXTENT:	DB		Extent
	0000	FCB\$RESV:	DB	0.0	Reserved for CP/M
000F		FCB\$RECUSED:	DB		Records used in this extent
		FCB\$ABUSED:	DB		0,0,0,0 ;Allocation blocks used
	0000000000		DB	0,0,0,0,	
0020		FCB\$SEQREC:	DB	0	:Sequential rec. to read/write
	0000	FCBSRANREC:	DW	õ	Random rec. to read/write
0023		FCB\$RANRECO:	DB	ŏ	;Random rec. overflow byte (MS)
0024	D204	RANRECNO:	DW	1234	;Example random record number
					Record will be written from ; address set by prior ; SETDMA call
0026	2A2400	LHLD	RANRECNO)	;Get random record number
0029	222100	SHLD	FCB\$RAN	REC	;Set up file control block
002C	0E28	MVI	C,B\$WRI	FERANZ	;Function code
002E	110000	LXI	D,FCB		;DE -> file control block
0031	CD0500	CALL	BDOS		;A = 00 if operation successful

```
;A = nonzero if no data in file
; specifically :
;A = 03 -- CP/M could not
; close current extent
; 05 -- directory full
; 06 -- attempt to write
; beyond end of disk
```

Purpose This function is an extension to the Write Random function described previously. In addition to performing the Write Random, it will also fill each new allocation block with 00H's. Digital Research added this function to assist Microsoft with the production of its COBOL compiler—it makes the logic of the file handling code easier. It also is an economical way to completely fill a random file with 00H's. You need only write one record per allocation block; the BDOS will clear the rest of the block for you.

Notes Refer to the description of the Write Random function (code 34).

The BIOS Components The BIOS Entry Points Bootstrap Functions Character Input/Output Functions Disk Functions Calling the BIOS Functions Directly Example BIOS



The Basic Input/Output System

This chapter takes a closer look at the Basic Input/Output System (BIOS). The BIOS provides the software link between the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the physical hardware of your computer system. The CCP and BDOS interact with the parts of your computer system only as logical devices. They can therefore remain unchanged from one computer system to the next. The BIOS, however, is customized for your particular type of computer and disk drives. The only predictable part of the BIOS is the way in which it interfaces to the CCP and BDOS. This must remain the same no matter what special features are built into the BIOS.

The BIOS Components

A standard BIOS consists of low-level subroutines that drive four types of physical devices:

- Console: CP/M communicates with the outside world via the console. Normally this will be a video terminal or a hard-copy terminal.
- "Reader" and "punch": These devices are normally used to communicate between computer systems—the names "reader" and "punch" are just historical relics from the early days of CP/M.
- List: This is a hard-copy printer, either letter-quality or dot-matrix.
- Disk drives: These can be anything from the industry standard single-sided, single-density, 8-inch floppy diskette drives to hard disk drives with capacities of several hundred megabytes.

The BIOS Entry Points

The first few instructions of the BIOS are all jump (JMP) instructions. They transfer control to the 17 different subroutines in the BIOS. The CCP and the BDOS, when making a specific request of the BIOS, do so by transferring control to the appropriate JMP instruction in this BIOS *jump table* or *jump vector*. The BIOS jump vector always starts at the beginning of a 256-byte page, so the address of the first jump instruction is always of the form xx00H, where "xx" is the page address. Location 0000H to 0002H has a jump instruction to the second entry of the BIOS jump vector—so you can always find the page address of the jump vector by looking in location 0002H.

Figure 6-1 shows the contents of the BIOS jump vector along with the page-relative address of each jump. The labels used in the jump instructions have been adopted by convention.

The following sections describe the functions of each of the BIOS's main subroutines. You should also refer to Digital Research's manual CP/M 2.0 Alteration Guide for their description of the BIOS routines.

Bootstrap Functions

There are two bootstrap functions. The cold bootstrap loads the entire CP/M operating system when the system is either first turned on or reset. The warm bootstrap reloads the CCP whenever a program branches to location 0000H.

**	OOH JMP	BOOT	;"Cold" (first time) bootstrap
	03H JMP	WBOOT	;"Warm" bootstrap
xx	OGH JMP	CONST	Console input status
××	09H JMP	CONIN	;Console input
××	OCH JMP	CONOUT	Console output
××	OFH JMP	LIST	:List output
××	12H JMP	PUNCH	;"Punch" output
××	15H JMP	READER	;"Reader" input
××	1SH JMP	HOME	;Home disk heads (to track 0)
××	1BH JMP	SELDSK	;Select logical disk
××	1EH JMP	SETTRK	;Set track number
XX	21H JMP	SETSEC	;Set sector number
××	24H JMP	SETDMA	;Set DMA address
××	27H JMP	READ	;Read (128-byte) sector
××	2AH JMP	WRITE	;Write (128-byte) sector
××	2DH JMP	LISTST	;List device output status
××	зон јмр	SECTRAN	Sector translate

Figure 6-1. Layout of the standard BIOS jump vector

BOOT: "Cold" Bootstrap

The BOOT jump instruction is the first instruction executed in CP/M. The bootstrap sequence must transfer control to the BOOT entry point in order to bring up CP/M. In general, a PROM receives control either when power is first applied or after you press the RESET button on the computer. This reads in the CP/M loader on the first sector of the physical disk drive chosen to be logical disk A. This CP/M loader program reads the binary image of the CCP, BDOS, and BIOS into memory at some predetermined address. Then it transfers control to the BOOT entry point in the BIOS jump vector.

This BOOT routine must initialize all of the required computer hardware. It sets up the baud rates for the physical console (if this has not already been done during the bootstrap sequence), the "reader," "punch," and list devices, and the disk controller. It must also set up the base page of memory so that there is a jump at location 0000H to the warm boot entry point in the BIOS jump vector (at xx03H) and a jump at location 0005H to the BDOS entry point.

Most BOOT routines sign on by displaying a short message on the console, indicating the current version of CP/M and the computer hardware that this BIOS can support.

The BOOT routine terminates by transferring control to the start of the CCP + 6 bytes (the CCP has its own small jump vector at the beginning). Just before the BOOT routine jumps into the CCP, it sets the C register to 0 to indicate that logical disk A is to be the default disk drive. This is what causes "A>" to be the CCP's initial prompt.

The actual CCP entry point is derived from the base address of the BIOS. The CCP and BDOS together require 1E00H bytes of code, so the first instruction of the CCP starts at BIOS -1E00H.

WBOOT: "Warm" Bootstrap

Unlike the "cold" bootstrap entry point, which executes only once, the WBOOT or warm boot routine will be executed every time a program terminates by jumping to location 0000H, or whenever you type a CONTROL-C on the console as the first character of an input line.

The WBOOT routine is responsible for reloading the CCP into memory. Programs often use all of memory up to the starting point of the BDOS, overwriting the CCP in the process. The underlying philosophy is that while a program is executing, the CCP is not needed, so the program can use the memory previously occupied by the CCP. The CCP occupies 800H (2048) bytes of memory—and this is frequently just enough to make the difference between a program that cannot run and one that can.

A few programs that are self-contained and do not require the BDOS's facilities will also overwrite the BDOS to get another 1600H (5632) bytes of memory. Therefore, to be really safe, the WBOOT routine should read in both the CCP and the BDOS. It also needs to set up the two JMPs at location 0000H (to WBOOT itself) and at location 0005H (to the BDOS). Location 0003H should be set to the initial value of the IOBYTE if this is implemented in the BIOS.

As its last act, the WBOOT routine sets register C to indicate which logical disk is to be selected (C = 0 for A, 1 for B, and so on). It then transfers control into the CCP at the first instruction in order to restart the CCP. Again, the actual address is computed based on the knowledge that the CCP starts 1E00H bytes lower in memory than the base address of the BIOS.

Character Input/Output Functions

Character input/output functions deal with logical devices: the console, "reader," "punch," and list devices. Because these logical devices can in practice be connected by software to one of several physical character I/O devices, many BIOS's use CP/M's IOBYTE features to assign logical devices to physical ones.

In this case, each of the BIOS functions must check the appropriate bit fields of the IOBYTE (see Figure 4-2 and Table 4-1) to transfer control to the correct physical device *driver* (program that controls a physical device).

CONST: Console Input Status

CONST simply returns an indicator showing whether there is an incoming character from the console device. The convention is that A = 0FFH if a character is waiting to be processed, A = 0 if one is not. Note that the zero flag need not be set to reflect the contents of the A register — it is the contents that are important.

CONST is called by the CCP whenever the CCP is in the middle of an operation that can be interrupted by pressing a keyboard character.

The BDOS will call CONST if a program makes a Read Console Status function call (B\$CONST, code 11, 0BH). It is also called by the console input BIOS routine, CONIN (described next).

CONIN: Console Input

CONIN reads the next character from the console to the A register and sets the most significant (parity) bit to 0.

Normally, CONIN will call the CONST routine until it detects A = 0FFH. Only then will it input the data character and mask off the parity bit.

CONIN is called by the CCP and by the BDOS when a program executes a Read Console Byte function (B\$CONIN, code 1).

CONOUT: Console Output

CONOUT outputs the character (in ASCII) in register C to the console. The most significant (parity) bit of the character will always be 0.

CONOUT must first check that the console device is ready to receive more data, delaying if necessary until it is, and only then sending the character to the device.

CONOUT is called by the CCP and by the BDOS when a program executes a Write Console Byte function (B\$CONOUT, code 2).

LIST: List Output

LIST is similar to CONOUT except that it sends the character in register C to the list device. It too checks first that the list device is ready to receive the character. LIST is called by the CCP in response to the CONTROL-P toggle for printer echo of console output, and by the BDOS when a program makes a Write Printer Byte or Display String call (B\$LISTOUT and B\$PRINTS, codes 5 and 9).

PUNCH: "Punch" Output

PUNCH sends the character in register C to the "punch" device. As mentioned earlier, the "punch" is rarely a real paper tape punch. In most BIOS's, the PUNCH entry point either returns immediately and is effectively a null routine, or it outputs the character to a communications device, such as a modem, on your computer.

PUNCH must check that the "punch" device is indeed ready to accept another character for output, and must wait if it is not.

Digital Research's documentation states that the character to be output will always have its most significant bit set to 0. This is not true. The BDOS simply transfers control over to the PUNCH entry point in the BIOS; the setting of the most significant bit will be determined by the program making the BDOS function request (B\$PUNOUT, code 4). This is important because the requirement of a zero would preclude being able to send pure binary data via the BIOS PUNCH function.

READER: "Reader" Input

As with the PUNCH entry point, the READER entry point rarely connects to a real paper tape reader.

The READER function must return the next character from the reader device in the A register, waiting, if need be, until there is a character.

Digital Research's documentation again says that the most significant bit of the A register must be 0, but this is not the case if you wish to receive pure binary information via this function.

READER is called whenever a program makes a Read "Reader" Byte function request (B\$READIN, code 3).

Disk Functions

All of the disk functions that follow were originally designed to operate on the 128-byte sectors used on single-sided, single-density, 8-inch floppy diskettes that were standard in the industry at the time. Now that CP/M runs on many different types of disks, some of the BIOS disk functions seem strange because most of the new disk drives use sector sizes other than 128 bytes.

To handle larger sector sizes, the BIOS has some additional code that makes the BDOS respond as if it were still handling 128-byte sectors. This code is referred to as the *blocking/deblocking* code. As its name implies, it blocks together several 128-byte "sectors" and only writes to the disk when a complete *physical* sector has been assembled. When reading, it reads in a physical sector and then deblocks it, handing back several 128-byte "sectors" to the BDOS.

To do all of this, the blocking/deblocking code uses a special buffer area of the same size as the physical sectors on the disk. This is known as the host disk buffer or HSTBUF. Physical sectors are read into this buffer and written to the disk from it.

In order to optimize this blocking/deblocking routine, the BIOS has code in it to reduce the number of times that an actual disk read or write occurs. A side effect is that at any given moment, several 128-byte "sectors" may be stored in the HSTBUF, waiting to be written out to the disk when HSTBUF becomes full. This sometimes complicates the logic of the BIOS disk functions. You cannot simply select a new disk drive, for example, when the HSTBUF contains data destined for another disk drive. You will see this complication in the BIOS only in the form of added logical operations; the BIOS disk functions rarely trigger immediate physical operations. It is easier to understand these BIOS functions if you consider that they make *requests*—and that these requests are satisfied only when it makes sense to do so, taking into account the blocking/deblocking logic.

HOME: Home Disk

HOME sets the requested track and sector to 0.

SELDSK: Select Disk

SELDSK does not do what its name implies. It does not (and must not) physically select a logical disk. Instead, it returns a pointer in the HL register pair to the disk parameter header for the logical disk specified in register C on entry. C = 0 for drive A, 1 for drive B, and so on. SELDSK also stores this code for the requested disk to be used later in the READ and WRITE functions.

If the logical disk code in register C refers to a nonexistent disk or to one for which no disk parameter header exists, then SELDSK must return with HL set to 0000H. Then the BDOS will output a message of the form

"BDOS Err on X: Select"

Note that SELDSK not only does not select the disk, but also does not indicate whether or not the requested disk is physically present —merely whether or not there are disk tables present for the disk.

SELDSK is called by the BDOS either during disk file operations or by a program issuing a Select Disk request (B\$SELDSK, code 14).

SETTRK: Set Track

SETTRK saves the requested disk track that is in the BC register pair when SETTRK gets control. Note that this is an absolute track number; that is, the number of reserved tracks before the file directory will have been added to the track number relative to the start of the logical disk.

The number of the requested track will be used in the next BIOS READ or WRITE function (described later in this chapter).

SETTRK is called by the BDOS when it needs to read or write a 128-byte sector. Legitimate track numbers are from 0 to 0FFFFH (65,535).

SETSEC: Set Sector

SETSEC is similar to SETTRK in that it stores the requested sector number for later use in BIOS READ or WRITE functions. The requested sector number is handed to SETSEC in the A register; legitimate values are from 0 to 0FFH (255).

The sector number is a logical sector number. It does not take into account any sector skewing that might be used to improve disk performance.

SETSEC is called by the BDOS when it needs to read or write a 128-byte sector.

SETDMA: Set DMA Address

SETDMA saves the address in the BC register pair in the requested DMA address. The next BIOS READ or WRITE function will use the DMA address as a pointer to the 128-byte sector buffer into which data will be read or from which data will be written.

The default DMA address is 0080H. SETDMA is called by the BDOS when it needs to READ or WRITE a 128-byte sector.

READ: Read Sector

READ reads in a 128-byte sector provided that there have been previous BIOS function calls to

SELDSK — "select" the disk

SETDMA-set the DMA address

SETTRK—set the track number

SETSEC-set the sector number.

Because of the blocking/deblocking code in the BIOS, there are frequent occasions when the requested sector will already be in the host buffer (HSTBUF), so that a physical disk read is not required. All that is then required is for the BIOS to move the appropriate 128 bytes from the HSTBUF into the buffer pointed at by the DMA address.

Only during the READ function will the BIOS normally communicate with the physical disk drive, selecting it and seeking to read the requested track and sector. During this process, the READ function must also handle any hardware errors that occur, trying an operation again if a "soft," or recoverable, error occurs.

The READ function must return with the A register set to 00H if the read operation is completed successfully. If the READ function returns with the A register set to 01H, the BDOS will display an error message of the form

BDOS Err on X: Bad Sector

Under these circumstances, you have only two choices. You can enter a CARRIAGE RETURN, ignore the fact that there was an error, and attempt to make sense of the data in the DMA buffer. Or you can type a CONTROL-C to abort the operation, perform a warm boot, and return control to the CCP.

As you can see, CP/M's error handling is not particularly helpful, so most BIOS writers add more sophisticated error recovery right in the disk driver. This can include some interaction with the console so that a more determined effort can be made to correct errors or, if nothing else, give you more information as to what has gone wrong. Such error handling is discussed in Chapter 9.

If you are working with a hard disk system, the BIOS driver must also handle the management of bad sectors. You cannot simply replace a hard disk drive if one or two sectors become unreadable. This bad sector management normally requires that a directory of "spare" sectors be put on the hard disk before it is used to store data. Then, when a sector is found to be bad, one of the spare sectors is substituted in its place. This is also discussed in Chapter 9.

WRITE: Write Sector

WRITE is similar to READ but with the obvious difference that data is transferred from the DMA buffer to the specified 128-byte sector. Like READ, this function requires that the following function calls have already been made:

SELDSK — "select" the disk SETDMA—set the DMA address SETTRK — set the track number SETSEC—set the sector number.

Again, it is only in the WRITE routine that the driver will start to talk directly to the physical hardware, selecting the disk unit, track, and sector, and transferring the data to the disk.

With the blocking/deblocking code, the BDOS optimizes the number of disk writes that are needed by indicating in register C the type of disk write that is to be performed:

0 = normal sector write

1 = write to file directory sector

2 = write to sector of previously unused allocation block.

Type 0 occurs whenever the BDOS is writing to a data sector in an already used allocation block. Under these circumstances, the disk driver must preread the appropriate host sector because there may be previously stored information on it.

Type 1 occurs whenever the BDOS is writing to a file directory sector — in this case, the BIOS must not defer writing the sector to the disk, as the information is too valuable to hold in memory until the HSTBUF is full. The longer the information resides in the HSTBUF, the greater the chance of a power failure or glitch, making file data already physically written to the disk inaccessible because the file directory is out of date.

Type 2 occurs whenever the BDOS needs to write to the first sector of a previously unused allocation block. Unused, in this context, includes an allocation block that has become available as a result of a file being erased. In this case, there is no need for the disk driver to preread an entire host-sized sector into the HSTBUF, as there is no data of value in the physical sector.

As with the READ routine, the WRITE function returns with A set to 00H if the operation has been completed successfully. If the WRITE function returns with A set to 01H, then the BDOS will display the *same* message as for READ: You can see now why most BIOS writers add extensive error-recovery and user-interaction routines to their disk drivers.

For hard disk systems, some disk drivers are written so that they automatically "spare out" a failing sector, writing the data to one of the spare sectors on the disk.

LISTST: List Status

As you can tell from its position in the list of BIOS functions, the LISTST function was a latecomer. It was added when CP/M was upgraded from version 1.4 to version 2.0.

This function returns the current status of the list device, using the IOBYTE if necessary to select the correct physical device. It sets the A register to 0FFH if the list device can accept another character for output or to 00H if it is not ready.

Digital Research's documentation states that this function is used by the DESPOOL utility program (which allows you to print a file "simultaneously" with other operations) to improve console response during its operation, and that it is acceptable for the routine always to return 00H if you choose not to implement it fully.

Unfortunately, this statement is wrong. Many other programs use the LISTST function to "poll" the list device to make sure it is ready, and if it fails to come ready after a predetermined time, to output a message to the console indicating that the printer is not ready. If you ever make a call to the BDOS list output functions, Write Printer Byte and Print String (codes 5 and 9), and the printer is not ready, then CP/M will wait forever — and your program will have lost control so it cannot even detect that the problem has occurred. If LISTST always returns a 00H, then the printer will always appear not to be ready. Not only does this make nonsense out of the LISTST function, but it also causes a stream of false "Printer not Ready" error messages to appear on the console.

SECTRAN: Sector Translate

SECTRAN, given a logical sector number, locates the correct physical sector number in the sector translate table for the previously selected (via SELDSK) logical disk drive.

Note that both logical and physical sector numbers are 128-byte sectors, so if you are working with a hard disk system, it is not too efficient to impose a sector interlace at the 128-byte sector level. It is better to impose the sector interlace right inside the hard disk driver, if at all; in general, hard disks spin so rapidly that CP/M simply cannot take advantage of sector interlace.

The BDOS hands over the logical sector number in the BC register pair, with the address of the sector translate table in the DE register pair. SECTRAN must return the physical sector number in HL.

If SECTRAN is to be a null routine, it must move the contents of BC to HL and return.

Calling the BIOS Functions Directly

As a general rule, you should not make direct calls to the BIOS. To do so makes your programs less transportable from one CP/M system to the next. It precludes being able to run these programs under MP/M, which has a different form of BIOS called an extended I/O system, or XIOS.

There are one or two problems, however, that can only be solved by making direct BIOS calls. These occur in utility programs that, for example, need to make direct access to the CP/M file directory, or need to access some "private" jump instructions which have been added to the standard BIOS jump vector.

If you really do need direct access to the BIOS, Figure 6-2 shows an example subroutine that does this. It requires that the A register contain a BIOS function code indicating the offset in the jump vector of the jump instruction to which control is to be passed.

	:	Equates	for use	e with BIOS subroutine
0003 =	WBOOT	EQU	озн	:Warm boot
0006 =	CONST	EQU	06H	:Console status
0009 =	CONIN	EQU	09H	;Console input
000C =	CONOUT	EQU	OCH	Console output
000F =	LIST	EQU	OFH	;Output to list device
0012 =	PUNCH	EQU	12H	;Output to punch device
0015 =	READER	EQU	15H	;Input from reader
0018 =	HOME	EQU	18H	Home selected disk to track O
001B =	SELDSK	EQU	1BH	;Select disk
001E =	SETTRK	EQU	1EH	;Set track
0021 =	SETSEC	EQU	21H	;Set sector
0024 =	SETDMA	EQU	24H	;Set DMA address
0027 =	READ	EQU	27H	;Read 128-byte sector
002A =	WRITE	EQU	2AH	;Write 128-byte sector
002D =	LISTST	EQU	2DH	;Return list status
0030 =	SECTRAN	EQU	30H	;Sector translate
	;			
				;Add further "private" BIOS codes here
	;			
	;	BIOS		
	;			e transfers control to the appropriate
	;			IOS Jump Vector, based on a code number
	;	handed	to it in	n the L register.
	;			
	;	Entry p	arametei	rs
	;			
	;	L = Code		r (which is in fact the page-relative
	;			s of the correct JMP instruction within
	;			np vector)
	;	All oth		sters are preserved and handed over to
	;		the BI	DS routine intact.
	;			
	;	Exit par	rameter	5
	-			

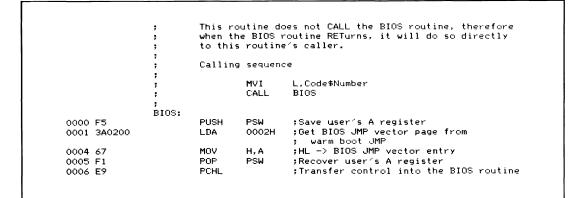


Figure 6-2. BIOS equates (continued)

Line Numbers	Functional Component or Routine
0072-0116	BIOS Jump Vector
0120-0270	Initialization Code
0275-0286	Display Message
0289-0310	Enter CP/M
0333-0364	CONST - Console Status
0369-0393	CONIN - Console Input
0397-0410	CONOUT - Console Output
0414-0451	LISTST - List Status
0456-0471	LIST - List Output
0476-0492	PUNCH - Punch Output
0496-0511	READER – Reader Input
0516~0536	IOBYTE Driver Select
0540-0584	Device Control Tables
0589-0744	Low-level Drivers for Console, List,etc.
0769-0824	Disk Parameter Header Tables
0831~0878	Disk Parameter Blocks
0881-0907	Other Disk data areas
0910-0955	SELDSK – Select Disk
0958-0964	SETTRK – Set Track
0967-0973	SETSEC – Set Sector
0978-0984	SETDMA – Set DMA Address
0987~1025	Sector Skew Tables
1028-1037	SECTRAN - Logical to Physical Sector translation
1041-1056	HOME - Home to Track O
1059-1154	Deblocking Algorithm data areas
1157-1183	READ - Read 128-byte sector
1185-1204	WRITE - Write 128-byte sector
1206-1378	Deblocking Algorithm
1381-1432	Buffer Move
1435-1478	Deblocking subroutines
1481-1590	8" Floppy Physical Read/Write
1595-1681	5 1/4" Floppy Physical Read/Write
1685-1764	WBOOT - Warm Boot

Figure 6-3. Functional Index to Figure 6-4

Example BIOS

The remainder of this chapter is devoted to an example BIOS listing. This actual working BIOS shows the overall structure and interface to the individual BIOS subroutines.

Unlike most BIOS's, this one has been written specifically to be understood easily. The variable names are uncharacteristically long and descriptive, and each block of code has commentary to put it into context.

Each source line has been sequentially numbered (an infrequently used option that Digital Research's Assembler, ASM, permits). Figure 6-3 contains a functional index to the BIOS as a whole so that you can find particular functions in the listing in Figure 6-4 by line number.

0001 < Line Numb	er ; Figure 6-4.	
0002 0003	;	
0003	;*************************************	
0005	;* * Simple BIOS Listing *	
0005	;* SIMPLE BLOS LISTING *	
0007	* *************************************	
0008	,	
0009		
0010 3030 =	, VERSION EQU '00' ;Equates used in the sign on m	
0011 3730 =	MONTH EQU '07'	lessage
0012 3531 =	DAY EQU 151	
0013 3238 =	YEAR EQU 1821	
0014		
0015	7 。 " 我说我说说我说我说我说我说我就是我说我说我说我说我说我说我说我说我说我说我说我说	*****
0016	*	*
0017	* This BIOS is for a computer system with the following	×
0018	;* hardware configuration ;	*
0019	**	*
0020	** - 8080 CPU	*
0021	* - 64KBytes of RAM	×
0022	;* - CRT/keyboard controller that transfers data	*
0023	** as though it were a serial port (but requires	*
0024	no baud rate generator or USART programming)	×
0025	* - A serial port, used for both list and "reader"/	×
0026	<pre>;* "punch" devices. The serial port chip is an</pre>	×
0027	i* Intel 8251A with an 8253 baud rate generator.	*
0028	;* - Two 5 1/4" mini-floppy, double-sided, double-	×
0029	# density drives. These drives use 512-byte sectors.	×
0030	;* These are used as logical disks A: and B:.	×
0031	;* - Two 8" standard diskette drives (128-byte sectors).	×
0032	7* These are used as logical disks C: and D:.	×
0033	ş ₩	*
0034	Fixed the second sec	×
0035	* each diskette type. These controllers access memory	×
0036	<pre>;* directly, both to read the details of the</pre>	×
0037	Fix operations they are to perform and also to read	×
0038	* and write data from and to the diskettes.	×
0039	; X	×
0040	ş x	×
0041	ş — — — — — — — — — — — — — — — — — — —	*****
0042		
0043	;	
0044	; Equates for defining memory size and the base address and	
0045	; length of the system components.	

0046												
0047	0040	-	, Memory\$Siz	2	EQU	64	;Number	r of Kl	bytes (of RAM		
0048			; : The BIO	5 Lenath	must he	determi	ned hv	inspec	tion.			
0050					ORG BIO					ing the	first	
0051			; characte	er to a s	semicolo	n. (This	: will ma	ake th	e Asser	nbler st	art	
0052			; the BIO	5 at loca	ation 0.) Then a	issemble	the B	IOS and	i round	up to	
0053					H the add	dress di	splayed	on th	e conso	ole at t	he end	
0054			; of the a	assembly.	•							
0055	0900	-	; BIOS\$Lengt		EQU	0900H						
0058	0900	-	BIUS#Lengti	1	EGO	07000						
0058	0800	=	, CCP\$Length	EQU	0800H	;Consta	int					
0059	0E00		BDOS\$Lengt		EQU		;Consta	ant				
0060			;									
0061	0008	=	Overall\$Le	ngth	EQU	((CCP\$L	.ength +	BDOS\$	Length	+ BIOS\$	(Length	/ 1024) + 1
0062	E000		; CCP\$Entry	500	/ Manau	•Ci	Overalls	*1				
0063	E806		BDOS\$Entry				overail: \$Length		n) * 10	24		
0065	F600		BIOSSEntry				\$Length		S\$Lengt	h		
0066			1									
0067			;									
0068			;									
0069	F/00		050	BIOCAE			10		ne			
0070	F600		ORG	BIOS\$En	try	; Assemb	le code	at BI	us addi	255		
0071			; ; BIOS ju	np vector	r							
0073					transfe	rred to	the app	ropria	te enti	y point		
0074			; from the	CCP or	the BDO	S, both	of which	COMP	ute the	e relati		
0075					BIOS jum							
0076					ams can							
0077				to loca	tion xx0	OH, when	e xx is	the v	alue in	n locati	on	
0078			; 0002H.									
0079	F600	C3F9F6	, JMP	BOOT	:Cold h	oot er	tered fi	rom CP	/M boot	strap 1	oader	
0081			Warm\$Boot\$				that the					
0082					; put	the warn	i boot ei	ntry a	ddress	down in	locatio	n
0083							02H of 1					
0084	F603	C329FE	JMP	WBOOT	Warm b	oot e	ntered t	oy jum	ping to) locati	on 0000H	•
0085							CCP which by prev:				sien‡	
0086						vritten ram area		ious p	JALEN	an treff	STELL	
0088	F606	C362F8	JMP	CONST			reti	urns A	= OFFH	l if the	re is a	
0089					; cons	ole keyb	oard cha	aracte	r waiti	ng		
0090	F609	C378F8	JMP	CONIN	;Consol	e input	retu	rns th	e next	console	keyboar	d
0091			 -		; char	acter in	A .					
0092	F60C	C386F8	JMP	CONOUT			out	outs t	he char	acter i	n U to	
0093	F/05	C34059	IMD		; the		device • output:	· the ·	obaraci	er in C	to the	
0094	FOUF	C3ACF8	JMP	LIST	; list		output	ุธ เกษา	characi	.er 111 C	, to the	
0095	F612	C3BCF8	JMP	PUNCH	Punch (output -	- output	ts the	charad	ter in	C to the	
0097							h device					
0098	F615	C3CDF8	JMP	READER	;Reader	input -	- returi	ns the		input ch	aracter	from
0099							reader d					
0100		C3D3FB	JMP	HOME			ently se					d
0101	F61B	C32BFB	JMP	SELDSK	1Select	s the di rne +h-	SK OFIVE	s SPec	iiled) o diek	n regis	ter C an er heade	r
0102	FAIF	C358FB	JMP	SETTRK							operati	
0103	1012	COUCH D	0.1		; from	the BC	register	r pair				
0105	F621	C35EFB	JMP	SETSEC					t read	or writ	e operat	ion
0106					; from							
0107	F624	C365FB	JMP	SETDMA	;Sets t	he dired	t memory	y addr	ess (di	sk read	/write)	
0108					; addr	ess for	the next	t read	or wri	te oper	ation	
0109	F(07	COEDER	IMD	DEAD			register		ied tr	ock and	sector f	rom
0110	F627	C3FBFB	JMP	READ			ldisk i				sector 1	
0111	F62A	C315FC	JMP	WRITE							sector	onto
0113	, Jan						disk fi					
0114	F62D	C394F8	JMP	LISTST	;Return						iccept	
0115							ut chara					
	F630	C3CDFB	JMP	SECTRAN	;Transl	ates a 🖯	ogical	sector	into a	a physic	al one	
0117			:									
0119			;									
0120				d boot in	nitializ	ation co	de is o	nly ne	eded or	nce.		
L												

0121 0122 0123 0124 0125			; It can be overwritten once it has been executed. ; Therefore, it is "hidden" inside the main disk buffer. ; When control is transferred to the BOOT entry point, this ; code will be executed, only being overwritten by data from ; the disk once the initialization procedure is complete.									
0126 0127 0128 0129 0130 0131 0132 0133 0134 0135			; ; To hide code in the buffer, the buffer is first declared ; normally. Then the value of the location counter following ; the buffer is noted. Then, using an ORG (ORiGin) statement, the ; location counter is "wound back" to the start of the buffer ; again and the initialization code written normally. ; At the end of this code, another ORG statement is used to ; set the location counter back as it was after the buffer had ; been declared.									
0136 0137 0138 0139 0140 0141	0200	=	; Ph	ysical\$S	ector\$Si	Ze	EQU	;for the ;The 8" ;Declare	;This is the actual sector size e 5 1/4" mini-floppy diskettes. diskettes use 128-byte sectors. e the physical disk buffer for the diskettes			
0142			Di ;	sk\$buffe	r:	DS	Physica	1\$Sectors	\$Size			
0144 0145 0146	F833	=	Af ;	ter\$Disk	\$Buffer	EQU	\$		he location counter rrent value of location counter			
0147	F633					ORG	Disk\$Bu	lfer	;Wind the location counter back			
0148			; In	itialize	\$Stream:				used by the			
0150							lize sub		It has the following			
0152						;		_				
0153						;	DB DB		mber to be initialized of bytes to be output			
0155						;	DB		of bytes to be output x,xx data to be output			
0156						;	:		•			
0157						;	: DB	Port num	mber of 00H terminator			
0159						;						
0160						;Note : ;			, the console port does initialized. This has			
0162						,			ne by the PROM bootstrap code.			
0163						;	;]niti~	lize the	8251A USART used for			
0165					_		; the	list and	communications devices.			
0166				DB		cation\$S	tatus\$Po	rt	;Port number			
0167	F634 F635			DB DB	6 0		:Get ch	ip ready	;Number of bytes to be programmed by			
0169	F636	00		DB	õ		; send	ing dummy	v data out to it			
0170	F637			DB	0	108						
0171	F638 F639			DB DB	0100\$00				e data terminal ready parity, 8 bits per character			
0173							; baud	l rate div	vide factor of 16.			
0174	F63A	25		DB	0010\$01	01B	;Raise	request t	to send, and enable			
0175			;				; tran	smit and	recelve.			
0177									8253 programmable interval			
0178							; time	r used to	generate the baud rate for			
0179	F63B	DF		DB	Communi	cation\$Ba		8251A USA	ART ;Port number			
0181	F63C	01		DB	1				Number of bytes			
0182	F63D	в6		DB	10\$11\$0	11\$0B			2, load LS byte first, baud rates), binary count.			
0184			;						Sada Fates/, Dinary COUNT.			
0185	F63E			DB		cation\$Ba	aud\$Rate		;Port number			
0186	F63F F640	02 3800		DB DW	2 0038H		;1200 b	aud (hace	;Number of bytes ed on 16X divide-down selected			
0188				- •				he 8251A				
0189	F/ **	00	;	DP	0							
0190	F642	00	;	DB	0		;rort n	umber of	O terminates			
0192			;	_								
0193			;	Equates	for the	sign-on	message					
	0000	=	ČR	EQU	ODH		;Carria	ge return	1			

000A	=	LF EQU :	OAH	;Line feed
		, Signon\$Me		;Main sign-on message
	43502F4D20		1CP/M 2.2.1	
F64C		DW	VERSION	;Current version number
F64E		DB DW	MONTH	-Compart data
F64F F651		DB	MUNTH	;Current date
F651		DB	DAY	
F654		DB	111	
F655		DW	YEAR	
	ODOAOA	DB	CR,LF,LF	
	53696D706C	DB	'Simple BIOS'	, CR, LF, LF
F668	4469736B20	DB	'Disk configu	ration :/.CR.LF.LF
	2020202020		A: 0.35	i Mbyte 5" Floppy',CR,LF
	2020202020		B: 0.35	Mbyte 5" Floppy',CR,LF,LF
	2020202020		C: 0.24	Mbyte 8" Floppy′,CR,LF Mbyte 8" Floppy′,CR,LF
F6DA	2020202020		D: 0.24	Mbyte 8" Floppy',CR,LF
E458		; DB	0	
F6F8	00		U U	
0004	-	; Default\$E	isk EQU	0004H ;Default disk in base page
0004		Sereuit≢L 1		AAAAA ADELGUIC DISK IN DASE ABAE
		, BOOT:	:Entered dire	ctly from the BIOS JMP vector.
				, be transferred here by the CP/M
			; bootstrap	
				zation state of the computer system
			; will be de	stermined by the
			•	strap and the CP/M loader setup.
			;	• • • • • •
				;Initialize system.
				;This routine uses the Initialize\$Stream
F6F9	E3	DI		; declared above. ;Disable interrupts to prevent any
r or 7	- 3	01		; side effects during initialization.
EKEA	2133F6	LXI	H, Initialize¶	Stream ;HL -> Data stream
1 OF H	2100,0	:	.,, .,,	versam yne y bata stream
		, Initializ	e\$Loop:	
F6FD	7E	MOV	A, M	;Get port number
F6FE		ORA	A	; If OOH, then initialization complete
	CA13F7	JZ	Initialize\$Co	omplete
	320AF7	STA		ort ;Set up OUT instruction
F705	23	INX	н	;HL -> Count of number of bytes to output
F706	4E	MOV	С,М	;Get byte count
		; 		
			e\$Next\$Byte:	THE STREET ALL ALL ALL ALL ALL ALL ALL ALL ALL AL
F707		INX MOV	H M	;HL -> Next data byte
F708 F709		MUV DB	A, M OUT	;Get next data byte ;Output to correct port —
F707		Initializ		, output to conject port
F70A		DB	0	;<- Set above
F70B		DCR	č	;Count down
F70C	C207F7	JNZ		
F70F	23	INX	н	;HL -> Next port number
	C3FDF6	JMP	Initialize\$Lo	oop ;Go back for next port initialization
		;		
			e\$Complete:	
		;		
F740	2501			
F713		MVI	A,00\$00\$00\$0 IOBYTE	
F/13	320300	STA	TOPLE	; is to act as console
F718	2143F6	LXI	H,Signon\$Mess	age ;Display sign-on message on consol
	CD33F8	CALL	Display\$Messa	
		;		
		•		
F71E		XRA	A	;Set default disk drive to A:
F71F	320400	STA	Default\$Disk	
F722	FB	EI		;Interrupts can now be enabled
		;		
F723	C340F8	JMP	Enter\$CPM	Complete initialization and enter
				; CP/M by going to the Console Command
		_		; Processor.
		; . End of		distinction and
		; End of	COIG BOOK 1819	tialization code
		;		

Figure 6-4. (Continued)

r					
0272 0273	F833		ORG	After\$Disk\$Buff	er ;Reset location counter
0274			;		
0275			Display\$Me	ssage: :Displa	ays the specified message on the console.
0276				:On ent	ry, HL points to a stream of bytes to be
0277				t outs	put. A OOH-byte terminates the message.
0278	F833	7F	MOV	A, M , 0017	;Get next message byte
0279	F834	B7	ORA	A	;Check if terminator
0280	F835		RZ	8	;Yes, return to caller
0281	F836		MOV	C.A	Prepare for output
0282	F837		PUSH	H H	;Save message pointer
0282		CD86F8	CALL	CONCUT	;Go to main console output routine
	F838		POP	H	;Recover message pointer
0284				н	
	F83C		INX		Move to next byte of message
0286	F83D	C333F8	JMP	Uisplay\$Message	;Loop until complete message output
0287			;		
0288			1		
0289			Enter\$CPM:	; This routine i	s entered either from the cold or warm
0290					It sets up the JMP instructions in the
0291					and also sets the high-level disk driver's
0292				; input/output	t address (also known as the DMA address).
0293			;		
	F840		MVI	A, JMP	;Get machine code for JMP
0295	F842	320000	STA	0000H	;Set up JMP at location 0000H
	F845	320500	STA	0005H	; and at location 0005H
0297			;		
		2103F6	LXI	H,Warm\$Boot\$Ent	
0299	F84B	220100	SHLD	0001H	;Put address at location 0001H
0300					
0301	F84E	2106E8	LXI	H,BDOS\$Entry	;Get BDOS entry point address
0302		220600	SHLD	6	;Put address at location 0005H
0303			;	-	
0304	F854	018000	LXI	B,80H	;Set disk I/O address to default
0305		CD65FB	CALL	SETDMA	;Use normal BIOS routine
0306		0000.0	;	02121M	yose normal proo roatine
0307	F85A	59	, EI		;Ensure interrupts are enabled
0308		3A0400	LDA	DefaulteDiel.	
0308	F85E		MOV	Default\$Disk	;Transfer current default disk to
			JMP	C,A	; Console Command Processor
0310	F80F	C300E0		CCP\$Entry	;Transfer to CCP
0311			;		
0312			;		
0313				input/output dri	vers
0314			;		
0315			; These c	lrivers all look	at the IOBYTE at location
0316			; 0003H,	which will have	been set by the cold boot routine.
0317			; The IOE	SYTE can be modif	ied by the STAT utility, by
0318			; BDOS ca	ills, or by a pro	gram that puts a value directly
0319			; into lo	cation 0003H.	
0320			;		
0321			; All of	the routines mak	e use of a subroutine, Select\$Routine,
0322					gnificant two bits of the A register
0323			; and use	s them to transf	er control to one of the routines whose
0324			; address	immediately fol	lows the call to Select\$Routine.
0325					elect\$Routine\$21, uses bits
0326					job this saves some space
0327			; by avoi	ding an unnecess	ary instruction.
0328			;		
0329	0003	=	IOBYTE	EQU 0003H	;I/O redirection byte
0330			;		·
0331			;		
0332			;		
0333			CONST:	;Get co	nsole status
0334					d directly from the BIOS JMP vector
0335					returns a parameter that reflects whether
0336					e is incoming data from the console.
0337				, the	t is incoming data from the composer
0338				A = 00	H (zero flag set) if no data
0338					FH (zero flag clear) if data
0340				,	IN VIEW IIAY CIEDIS II UALA
0340				i CONCT	will be ealled by programs that
0341				; CONST	will be called by programs that Periodic checks to see if the computer
0342					ator has pressed any keys for example,
0343					
				; to i	nterrupt an executing program.
0345			 .	0-100 1	the state of the s
	F862	CD6AF8	CALL	Get\$Console\$Sta	
0347					According to status, then convert
1					

Figure 6-4. (Continued)

F865 B7	ORA	A	; to return parameter convention. ;Set flags to reflect status
F866 C8	RZ		; Jet flags to reflect status ; If O, no incoming data
F867 3EFF	MVI	A, OFFH	;Otherwise return A = OFFH to
F869 C9	RET		; indicate incoming data
	;		
F86A 3A0300	Get\$Consc LDA	le\$Status: IOBYTE	;Get I/O redirection byte
·86A 3A0300	LUA	TOBALE	;Get I/U redirection byte ;Console is selected according to
			; bits 1,0 of IOBYTE
86D CDDCF8	CALL	Select\$Routine	;Select appropriate routine
			These routines return to the caller
			<pre>; of Get\$Console\$Status.</pre>
F870 F6F8	DW	Teletype\$In\$Sta	
-872 FCF8	DW	Terminal\$In\$Sta	
-874 02F9 -876 08F9	DW DW	Communication\$I	n\$Status ;10 :11
-0/6 U8F9	:	Dummy\$In\$Status	711
	;		
	;		
	;		
	CONIN:		nsole input character
			d directly from the BIOS JMP vector;
			rns the next data character from the
		; Cons	ole in the A register. The most significant
			of the data character will be 0, except "reader" (communication port) input has
			selected. In this case, the full eight bits
			ata are returned to permit binary data to be
			ived.
		;	
			ly, this routine will be called after
			11 to CONST has indicated that a data character
		; is r	ady, but whenever the CCP or the BDOS can
			eed no further until console input occurs, CONIN will be called without a preceding
			Conin will be called without a preceding F call.
		; :::::::::::::::::::::::::::::::::::::	
878 3A0300	LDA	IOBYTE	;Get I/O redirection byte
F87B CDDCF8	CALL		Select correct CONIN routine
			;These routines return directly
			; to CONIN's caller.
F87E 20F9	DW	Teletype\$Input	;00 <- IOBYTE bits 1,0
-880 26F9	DW	Terminal\$Input	;01
F882 2FF9 F884 35F9	DW DW	Communication\$I Dummy\$Input	nput ;10 ;11
·884 33F9	:	Dawmy#Input	ş11
	;		
	;		
	CONOUT:	;Consol	e output
			d directly from BIOS JMP vector;
		; outpu	ts the data character in the C register
		; to th	e appropriate device according to bits
			f IOBYTE
F886 3A0300	LDA	; IOBYTE	;Get I/O redirection byte
F889 CDDCF8	CALL	Select\$Routine	;Select correct CONOUT routine
	Unit	Selectenoutlie	; These routines return directly
-887 CDDC-8			; to CONOUT's caller.
-887 CDDCF8	nw	Teletype\$Output	;00 <- IOBYTE bits 1,0
		Terminal\$Output	;01
F88C 38F9 F88E 3EF9	DW		
F88C 38F9 F88E 3EF9 F890 44F9	DW DW	Communication\$0	
F88C 38F9	DW DW DW	Communication\$0 Dummy\$Output	;11
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ;		;11
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ; ;		;11
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ; ;	Dummy\$Output	
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ; ;	Dummy\$Output ;List d	evice (output) status
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ; ;	Dummy\$Output ;List d ;Entere ; retur;	evice (output) status d directly from the BIOS JMP vector; s in A list device status that
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ; ;	Dummy\$Output ;List d ;Entere; ; retur; ; indic.	evice (output) status d directly from the BIOS JMP vector; ns in A list device status that ates whether the list device can accept
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ; ;	Dummy\$Output ;List d ;Entere ; retur ; indic ; anoth	evice (output) status d directly from the BIOS JMP vector; hs in A list device status that ales whether the list device can accept er output character. The IOBYTE's bits
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ; ;	Dummy\$Output ;List d ;Entere ; retur ; indic ; anoth	evice (output) status d directly from the BIOS JMP vector; ns in A list device status that ates whether the list device can accept
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ; ;	Dummy\$Output ;List d ;Entere ; retur ; indic ; anoth ; 7.6 d	evice (output) status d directly from the BIOS JMP vector; hs in A list device status that ales whether the list device can accept er output character. The IOBYTE's bits etermine the physical device used.
F88C 38F9 F88E 3EF9 F890 44F9	DW DW DW ; ;	Dummy\$Output ;List d ;Entere; ; retur ; indic ; anoth ; 7,6 d ; ; A = 00	evice (output) status d directly from the BIOS JMP vector; hs in A list device status that ales whether the list device can accept er output character. The IOBYTE's bits

0424 IDisital Research's documentation indicates 0425 ithat you can always return with A = OOH 0426 implement the LIST routine. This NOT TWC. 0427 implement the LIST routine. This NOT TWC. 0428 implement the LIST routine. This is NOT TWC. 0429 implement the LIST routine. This is NOT TWC. 0430 implement the LIST routine. This is NOT TWC. 0431 respective. 0432 implement the LIST routine. This is NOT TWC. 0433 respective. 0434 respective. 0435 respective. 0436 respective. 0437 respective. 0438 respective. 0439 respective. 0439 respective. 0439 respective. 0439 respective. 0439 respective. 0431 respective. 0432 respective. 0439 respective. 0439 respective. 0449 respective. 0440 respective. 0441 respective.							
0431 , than polentially hanging the system. 0432 F894 CD9CE8 CALL Get#ListStatus ;Return A = zero or nonzero 0433 F897 AC D9CE8 CALL Get#ListStatus ;Return A = zero or nonzero 0435 F897 B7 ORA A isortines convention 0436 F897 B7 ORA A isortines convention 0437 F895 CB RZ iff 0. cannot accept data for output 0448 F895 CB RZ indicate can accept data for output 0444 F895 CA0300 CALL Select#Routine ;Select apropriate routine 0444 F896 CD CPC CALL Select#Routine ;Select apropriate routine 0444 F896 CD CPC CALL Select#Routine ;Select apropriate routine 0447 F896 CD CPC CALL Select#Routine ;Select apropriate routine 0448 F896 CD CPC CALL Select#Routine ;Select apropriate routine 0447 F896 CD CPC CALL Select#Routine ;Select aprocies routine 0448 F896 CD CPC CALL Select#Routine ;Select aprocies routine 0449 F896 CD CPC CALL Select#Routine ;Select correct LIST r	0425 0426 0427 0428 0429				; that ; ("Can ; imple ;If you ; alway	you can inot acce ment the i do not is return	always return with A = 00H pt data") if you do not wish to : LISTST routine. This is NOT TRUE. wish to implement the LISTST routine o with A = 0FFH ("Can accept data").
0433 F394 CD9CF8 CALL GetsListStatus; ;Return A = zero or nonzero 0433 F395 CD CALL GetsListStatus; ;Return A = zero or nonzero 0434 F395 CD RZ If 0, cannot accept data for output 0437 F395 CD RZ If 0, cannot accept data for output 0437 F395 CD RET y indicate can accept data for output 0438 F395 CD RET y indicate can accept data for output 0441 GettListStatus; Identify investor; A to 10 identify investor; A to 10 0442 F395 CD RLC iff vice bits 7.6 to 1.0 0443 F695 CD RLC select#Routine ;select appropriate routine 0444 F695 CD CALL Select#Routine ;select appropriate routine 0445 F695 CD CALL Select#Routine ;select appropriate routine 0446 F84 OBF9 DW Teletype80ut\$Status 101 0447 F84 OBF9 DW Teletype80ut\$Status 101 0448 F84 OBF9 DW Teletype80ut\$Status 101 0449 F84 OBF9 DW Teletype80ut\$Status 101					;The LI : than	ST drive potentia	r will then take care of things rather Illy hanging the system.
0433 is according to status, then convert 0435 is to return parameter convention 0435 FB97 ED ORA 0435 FB97 ED ORA 0435 FB97 ED ORA 0435 FB97 ED ORA 0436 FB97 ED ORA 0437 FB97 ED ORA 0438 FB97 ED ORA 0440 indicate can accest data for output 0441 Development indicate can accest data for output 0442 FB97 OO RE indicate can accest data for output 0444 FB97 OO RE indicate can accest data for output 0444 FB97 OO RE indicate can accest data for output 0444 FB97 OO RE Call DBVTE idet for output 0444 FB97 OO RE Call DBVTE idet for output 0445 FB41 ODF DW TeletypeOutStatus idet for output 0446 FB97 DW DW TeletypeOutStatus idet for output 0447 FB44 OF DW DWmmy4OutStatus idet for outpu	0432	5004 CD	00000	CAL 1	;		
0437 F898 CB RZ iff 0, cannot accept data for output 0438 F898 CF RT indicate can accept data for output 0439 F898 CF RT indicate can accept data for output 0441 GetLittStatus: GetLittStatus: GetLittStatus: 0442 F897 CF REC iMove bits 7,6 to 1.0 0444 F800 OF RLC iMove bits 7,6 to 1.0 0444 F804 OEFS DW Terminal@OutStatus iOC - IOBYTE bits 1.0 0445 F840 OEFS DW Terminal@OutStatus iOC - IOBYTE bits 1.0 0445 F840 OEFS DW Terminal@OutStatus iOC - IOBYTE bits 1.0 0445 F840 OEFS DW Terminal@OutStatus iII 0445 F840 IFFS DW Dummy@OutStatus iII	0434 0435				Get¥List¥Status	; accor	ding to status, then convert
0439 F895 3EFF NUI A. OFFH ; indicate can accest data for output 0440 ; indicate can accest data for output ; indicate can accest data for output 0441 ; indicate can accest data for output 0442 F895 30300 LBC indicate can accest data for output 0441 F895 30300 LBC indicate can accest data for output 0442 F895 30300 LBC indicate can accest data for output 0444 F895 30300 LBC indicate can accest data for output 0445 F846 OBFO RET ; idet indicate can accest data for output 0446 F846 OBFO RET ; idet indicate can accest data for output 0446 F846 OFFO RET ; idet indicate can accest data for output 0447 F846 OFFO RET idet indicate can accest data for output 0447 F846 OFFO RET idet indicate can accest data for output 0447 F846 IFFP DH Teats propriate device according to bits ; idet indicate can accest in the Cregister 0455					A		
0440 ; idetilist\$Status; ; Det 1/0 redirection byte 0442 FSPC 30300 LDA IDBYTE ; Det 1/0 redirection byte 0443 FSPC 07 RLC ; Howe bits 7.6 to 1.0 0444 FSPC 10007B CALL Select\$Provines return directly 0445 FSA4 0BF9 DW Teletype\$Out\$Status ; 100 0445 FSA4 0BF9 DW Terminal@Out\$Status ; 100 0445 FSA4 0BF9 DW Terminal@Out\$Status ; 100 0455 FSA4 1DF9 DW Terminal@Out\$Status ; 100 0451 FSA4 0BF9 DW Terminal@Out\$Status ; 100 0452 FSA6 1JF9 DW Dummy\$Out\$Status ; 100 0453 ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	0438	F899 3E	FF	MVI	A, OFFH	;Otherw	vise return A = OFFH to
0441 Get8List\$Status: 0442 F89F 07 RLC iNove bits 7.6 to 1.0 0444 F80A 027 RLC iNove bits 7.6 to 1.0 0445 F80A 027 CALL Select\$Prove bits 7.6 to 1.0 0446 F80A 027 CALL Select\$Prove but\$Status :00 0447 F80A 027 CALL Select\$Prove but\$Status :00 0448 F80A 027 DW Terminal@out\$Status :00 COUNT 0449 F80A 027 DW Terminal@out\$Status :10 0455		F89B C9	1	RET		; indic	ate can accept data for output
0443 F89F 07 RLC iMove bits 7.6 to 1.0 0444 F8A1 CDDCF8 CALL Select#Routine ;Select appropriate routines return directly 0445 F8A1 CDDCF8 CALL Select#Routine ;Select appropriate routines 0447 F8A4 OBF7 DW Teletype\$Cut\$Status 100 <- 108YTE bits 1.0	0441			, Get\$List\$S	Status:		
0444 F8A0 07 RLC 0445 F8A4 0BF9 DW Teletype&Out&Status ;10 GetBlist&Status's caller. 0446 F8A4 0BF9 DW Teletype&Out&Status ;00 < 108YTE bits 1.0					IOBYTE		
0445 F8A1 CDDCF8 CALL Select#Routine ;Select appropriate routines return directly ; These routines return directly ; to Get#List85tatus's caller. 0447 ;These routines return directly ; to Get#List85tatus's caller. 0448 F8A4 0BF? DW 0449 F8A6 1IF? DW 0445 F8A4 1DF? DW 0450 F8A4 1DF? DW 0451 F8A4 1DF? DW 0453 ; 0454 F8A4 1DF? DW 0455 ;						;Move b	its 7,6 to 1,0
0447 ito Get#Listestatus: scaller. 0448 F8A4 0FF9 DW Teletype#Out#Status ;00 <- IOBVTE bits 1.0	0445				Select\$Routine	;Select	
0449FRA4 OBF9DWTeletype0ut\$Status:00:00:00:000449FRA6 1F9DWCommunication\$Out\$Status:100451FRAA 1F9DWDummyfOut\$Status:110452;;::110453;;:110454;;:110455;;:110456LIST:;List output0457; outputs the data character in the C register0458;; outputs the data character in the C register0459; outputs the data character in the C register0450; outputs the data character in the C register0451; outputs the data character in the C register0452; outputs in the data character in the C register0453; outputs in the data character in the C register0454; outputs in the data character in the C register0455; outputs in the character in the C register0464; fB80 07RLC; Move bits 7.6 to 1.00465FB81 CDDCF80466; fB84 38F9DWTeletype0utput0470; fB88 44F9DWDummyfOutput0471; fB86 38F9DWCommunication\$0utput0472; fD88 44F9DWDummyfOutput0473; fD88 44F9DWDummyfOutput0474; fD88 44F9DWDummyfOutput0475; fD88 44F9OH47; fD88 44F9OH4							
0450FBA8 17F9DWCommunication#Out#Status;100451FBA8 1DF9DWDummy#Out#Status;110452;0453;0454;0455;0456LIST:;List output;Entered directly from BIOS_JMP vector;; outputs the data character in the C register; outputs the data character in the C register; outputs the data character in the C register; outputs the data character in the C register04620463046404640464046504640465046404650465046404650466046704680468046904690469046904690469046904690469046904600470047004710471047204730473047404740475047504760477047804790489048004810482048204830484048004830484048004830484048004830484048404840485 </td <td></td> <td>F8A4 OB</td> <td>F9</td> <td>DW</td> <td>Teletype\$Out\$St</td> <td>atus</td> <td></td>		F8A4 OB	F9	DW	Teletype\$Out\$St	atus	
Od51FBAA 1DF9DWDummy#Out#Status;11Od53;Od54;Od55;Od56LIST:;List outputOd57istered directly from BIOS JMP vector;Od58; outputs the data character in the C registerOd59; outputs the data character in the C registerOd59; 7,6 of IOBYTEOd60; 7,6 of IOBYTEOd61FAA 070Od62FAA 070Od63FBA 070RLC; for the appropriate device according to bitsOd64FBA 070RLC; for the percent list routineOd64FBA 38F9DWTeletype#OutputOd65FBB4 38F9DWTerminal@outputOd70FBB8 44F9DWDummy#OutputOd71;Od72;Od73;Od74;Od75;Od76PUNCH:; Funch output; outputs the data character in the C register; to the appropriate device according to bits; 5,4 of IOBYTEOd75;Od76RC; butputs the data character in the C register; to the appropriate device according to bits; 5,4 of IOBYTE; outputs the data character in the C register; to the appropriate device according to bits; 5,4 of IOBYTEOd77; cet I/O redirection byte; cet I/O redirection byte; cet I/O redirection byte; cet I/O							
0452 0453 ; 0454 ; 0455 ; 0456 LIST: ;List outputs the data character in the C register 0457 ; outputs the data character in the C register 0457 ; outputs the data character in the C register 0459 ; outputs the data character in the C register 0460 ; 7.6 of IOBYTE 0461 ; 7.6 of IOBYTE 0462 FBAC 3A0300 LDA 0464 FBB 007 RLC 0465 FSBE 1CDDCF8 CALL 0466 : These routines return directly 0467 FSBE 38F9 DW 0468 FSBE 38F9 DW 0471 FSBE 34F9 DW 0472 ; Dummy\$Output ;01 0473 ; outputs the data character in the C register ; 10877 PUNCH: ;Punch output ;01 0473 ; outputs the data character in the C register ; 10877 ;Det I/O redirection byte ; ; 0488 FSBE 3AF9 DW	0450						
0454;0455LIST:;List output0457;Entered directly from BIOS UMP vector;0459; outputs the data character in the C register0459; to the appropriate device according to bits0450; 7,6 of IOBYTE0461; 7,6 of IOBYTE0462FBAC 3A03000453FBB 070463FBB 070464FBB 070464FBB 070465FBB 1 CDDCF80466CALLSelect correct LIST routine0467; These routines return directly0468FBB4 38F90469FBB4 38F90469FBB4 38F90469FBB4 38F90469FBB4 34F90469FBB4 34F90469FBB4 34F9041Communication 90utput0421;00 (~ 10BYTE bits 1.00433;0444;0455;0456FBB4 34F904Communication 90utput0457;0457;0473;0474;100;0475;0476;0477;0478;0480;0479;0471;100;0472;0473;0474;1010475;102;103;104;104;<	0452						
0455;;List output0456LIST:;Entered directly from BIOS JMP vector;0458; outputs the data character in the C register0459; outputs the data character in the C register0450; to the appropriate device according to bits0461; 7,6 of IDBYTE0462FRAF 070463FRAF 070464FBB0 070465FRB1 CDDCFB0464CALL0465Select correct LIST routine0466; These routines return directly0467; to LIST's caller.0468FRB4 38F90470FRB4 38F90470FRB4 4AF90471FRB4 4AF90472;0473;0474;0475;0476PUNCH;; Funch output; to the appropriate device according to bits0473;0474;0475;0481;0482FRBC 3A03000483FRBF OF0481;0482FRBF OF0483FRBF OF0484;0485FRSC 0F0485FRSC 0F0486FRC0486FRSC 2CDDDFB0486FRSC 4AF50486FRSC 4AF50486FRSC 4AF50486FRSC 4AF50490FRSC 4AF50491FRSC 4AF50492;0493;1000FBCALL<							
0457;Entered directly from BIOS JMP vector;0458; outputs the data character in the C register0459; to the appropriate device according to bits0460; 7,6 of IOBYTE0461;0462FBAC 3A0300LDA0463FBAF 07RLC0464; BEB0 070465FBB1 CDDCF8CALL0466select correct LIST routine0467; BB4 38F90468FBB4 38F90469FBB6 3EF90470FBB6 3EF90471FBB6 44F90472;0473;0474;0475;0476PUNCH:; Funch output; outputs the data character in the C register; to the appropriate device according to bits; 5,4 of IOBYTE; 04820482FBBC 3A0300LDA10BYTE; 0404; 0476; 0477; 0478; 0478; 0479; 0474;; 0474;; 0475; 0476; 0476; 0477; 0478; 0480; 0481; 0481<	0455			;			
0458i outputs the data character in the C register0459; to the appropriate device according to bits0460; 7,6 of IOBYTE0461; to the appropriate device according to bits0462FBAC 3A0300LDA0463FBAF 07RLC0464FBB0 07RLC0465FBB1 CDDCF8CALL0466Select correct LIST routine0467i to LIST's caller.0468FBB4 38F9DW0470FB85 34F9DW0471FB88 44F9DW0472FB84 44F9DW0473;0474;0475;0476PUNCH:;Punch output0477;outputs the data character in the C register0478;outputs the data character in the C register0479;bunch:;Punch output0471;intered directly from BIOS JMP vector;0472;outputs the data character in the C register0473;0474;0475;0476PUNCH:;intered directly from BIOS JMP vector;0477;outputs the data character in the C register0478;outputs the data character0479;bunch:0470;bunch:0471;intered directly from BIOS JMP vector;0472;outputs the data character in the C register0473;0474;intered directly from BIOS JMP vector;0475;0476PUNCH:;int				LIST:			
0459 ; to the appropriate device according to bits 0460 ; 7,6 of IDBYTE 0461 ; 0462 FBAC 3A0300 LDA 0464 FBAC 70 RLC 0465 FBBD 07 RLC 0464 FBBD 07 RLC 0465 FBB1 CDDCF8 CALL 0466 FBB5 3BF9 DW 0467 FBB6 3EF9 DW 0468 FBB6 3EF9 DW 0471 FBB5 44F9 DW 0472 j Communication\$Output 0473 ; 0474 ; 0473 ; 0474 ; 0475 ; 0476 PUNCH; ;Punch output joutputs the appropriate device according to bits 0476 ; joutputs 0477 ; joutputs 0478 ; joutputs 0479 ; outputs the appropriate device according to bits 0483 FBBF OF RRC 0484 FBC 3A0300 <td< td=""><td></td><td></td><td></td><td></td><td>;Entere ; outpu</td><td>ts the d</td><td>ata character in the C register</td></td<>					;Entere ; outpu	ts the d	ata character in the C register
0461; Get I/O redirection byte0462FRAC 3A0300LAIDBYTE;Get I/O redirection byte0463FRAC 3A0300LAIDBYTE;Hove bits 7,6 to 1,00464FBB0 07RLC:Hove bits 7,6 to 1,00465FSB1 CDDCF8CALLSelect\$Routine;Select correct LIST routine0466rbss <rd>rbssrbss<rd>rbssrbss0467FSB4 38F9DWTeletype\$Output;000468FSB4 38F9DWTeletype\$Output;100470FSB8 44F9DWCommunication\$Output;110471FSBA 44F9DWCommunication\$Output;110472</rd></rd>					; to th	e approp	riate device according to bits
0462F8AC 3A0300LDAIOBYTE;Get I/O redirection byte0463F8AF 07RLC;Move bits 7,6 to 1,00464F8B0 07RLC;These routines return directly0465F8B1 CDDCF8CALLSelect\$Routine;These routines return directly0466;These routines return directly;0 to LIST's caller.0467;D488 44F9DWTerminal\$Qutput;010470F8B8 44F9DWCommunication\$Qutput;110471F8BA 4AF9DWDummy\$Qutput;110472;;0473;0474;0475;0476PUNCH:;Punch output;010477;0478;0479;0476;0477;0478;0479;0476;0477;0478;0479;0470;0471;0472;0473;0474;0475;0477;0478;0479;0480;<						I IOBYTE	
0464F8B0 07FLC0465F8B1 CDDCF8CALLSelect#Routine;Select correct LIST routine0466;These routines return directly;to LIST's caller.04670467;to LIST's caller.0468F8B4 38F9DWTerminal#Output0470F8B8 44F9DWCommunication#Output0471F8B4 4AF9DWDummy#Output0472;;0473;;0474;;0475;;0476PUNCH:;Punch output0477;outputs the data character in the C register0478; 5,4 of IOBYTE;Get I/O redirection byte0480;5,4 of IOBYTE;Get I/O redirection byte0481;;Get I/O redirection byte0483F8BF OFRRC0484F8C2 CDDF8CALLSelect#Routine#21;Select correct PUNCH routine0488;;0489F8C5 38F9DW0490F8C7 44F9DW0492F8C8 3EF9DW0494;0495;0496READER:; returnal#Output;110497;0498;0498;0499;0499;0490;0491;0492;0493;0494;0494;0495;0496READER:; input	0462		0300			;Get I/	O redirection byte
0465 0466FSB1 CDDCF8CALLSelect\$Routine;Select correct LIST routine ;These routines return directly ; to LIST's caller.0466 0467DWTeletype\$Output;00 <- IOBYTE bits 1,0						;Move b	its 7,6 to 1,0
0466; These routines return directly ; to LIST's caller.0468F8B4 38F9DW0469F8B6 3EF9DW0470F8B8 44F9DW0471F8B8 44F9DW0473;0474;0475;0476PUNCH:0477; Entered directly from BIOS JMP vector; ; outputs the data character in the C register ; to the appropriate device according to bits0480; 5,4 of IOBYTE ; Nove bits 5,4 to 2,10483F8EC 3A0300LDA0484F8C0 OF ; CDDF8CALL0485SE2 CDDDF8CALL Select \$RC0486; These routines return directly ; outputs the data charact print directly ; to the appropriate device according to bits ; there directly from BIOS JMP vector; ; to the directly from			DCF8		Select\$Routine		Select correct LIST routine
0468FBB4 38F9DWTeletype\$Output;00 <- IOBYTE bits 1,00469F8B6 3EF9DWTerminal\$Output;010470F8B8 44F9DWCommunication\$Output;100471F8B8 4AF9DWDummy\$Output;110472;							;These routines return directly
0469F886 3EF9DWTerminal\$Output:010470F888 44F9DWCommunication\$Output:100471F884 44F9DWDummy\$Output:110472		F8B4 38	F9	nω	Teletype\$Output		
0471F8BA 4AF9DWDummy\$Output;110472;0473;0474;0475;0476PUNCH:;Punch output0477; intered directly from BIOS JMP vector;0478; outputs the data character in the C register0479; outputs the data character in the C register0480; 5,4 of IOBYTE0481;0482F8BC 3A03000484F8C0 OF0485F8C1 OF0486F8C2 CDDDF80487; to elect\$Routine\$210488; to PUNCH's caller.0489F8C5 38F90WTeletype\$Output0490; outputs0491F8C9 44F90WCommunication\$Output0493;0494;0494;0495;0496READER:; inputs the next data character from the	0469	F8B6 3E	F9	DW	Terminal\$Output		;01
0472 0473 ; 0474 ; 0475 ; 0476 PUNCH: ;Punch output 0477 ; outputs the data character in the C register 0478 ; outputs the data character in the C register 0479 ; outputs the data character in the C register 0479 ; outputs the data character in the C register 0479 ; outputs the data character in the C register 0470 ; foutputs the data character in the C register 0470 ; foutputs the data character in the C register 0470 ; foutputs the data character in the C register 0470 ; foutputs the data character in the C register 0470 ; foutputs the data character in the C register 0470 ; foutputs the data character in the C register 0470 ; foutputs ; foutput						utput	
0474;0475;0476PUNCH:;Entered directly from BIOS JMP vector;0477;Entered directly from BIOS JMP vector;0478; outputs the data character in the C register0479; to the appropriate device according to bits0480; 5,4 of IOBYTE0481;0482 F8BC 3A0300LDA108YTE;Get I/O redirection byte;;0483 F8BF OFRRC0484 F8C0 OFRRC0485 F8C1 OFRRC0486 F8C2 CDDDF8CALLSelect\$Routine\$21;Select correct PUNCH routine0487; to PUNCH's caller.0488; to PUNCH's caller.0490 F8C7 4AF9DWDWDummy\$Output0491 F8C9 44F9DWOWTerminal\$Output0493;;; Reader input0493;;; Reader input0493;;; Inputs the next data character from the	0472		• •	2	Dammy +oatpat		,
0475;0476PUNCH:;Punch output0477;Entered directly from BIOS JMP vector;0478; outputs the data character in the C register0479; to the appropriate device according to bits0480; 5,4 of IOBYTE0481;0482F8BC 3A03000483F8BF OF0484F8BC 0FRRC; Get I/D redirection byte0485F8C1 OF0486F8C2 CDDDF80487CALLSelect\$Routine\$21; Select correct PUNCH routine0488; to PUNCH's caller.0489F8C5 38F9DWTeletype\$Output0490; 00 <- 10BYTE bits 1,0							
0477;Entered directly from BIOS JMP vector;0478; outputs the data character in the C register0479; to the appropriate device according to bits0480; 5,4 of IOBYTE0481; 5,4 of IOBYTE0482F8BC 3A0300LDA0483F8BF OFRRC0484F8C0 OFRRC0485F8C1 OFRRC0486F8C2 CDDDF8CALL0487; Select correct PUNCH routine0488; to PUNCH's caller.0489F8C5 38F9DW0491F8C9 44F9DW0493;0493;0493;0494;0495;0496READER:; inputs the next data character from the				;			
0478; outputs the data character in the C register0479; to the appropriate device according to bits0480; 5,4 of IOBYTE0481;0482F8BC 3A03000483F8BF OF0484F8C0 OF0484F8C1 OF0485F8C2 CDDDF80486F8C2 CDDDF80487; to PUNCH's caller.0489F8C5 38F90490Teletype\$Output0490; outputs the data character from the0491F8C9 44F90492; Recl0493;0494;0494;0495;0496READER:; inputs the next data character from the				PUNCH:			
0479 ; to the appropriate device according to bits 0480 ; 5,4 of IOBYTE 0481 ; 0482 F8BC 3A0300 LDA 0484 F8BC 9 0482 F8BC 3A0300 LDA 0483 F8BF 0F RRC 0484 F8C0 0F RRC 0485 F8C1 0F RRC 0486 F8C2 CDDDF8 CALL 0487 ;These routines return directly 0488 ;These routines return directly 0487 ;to PUNCH's caller. 0488 ;to PUNCH's caller. 0490 F8C5 38F9 DW 0490 F8C7 4AF9 DW 0491 F8C9 44F9 DW 0492 ;accommunicationsOutput ;10 0493 ; 04 0493 ; ; 0493 ; ; 0493 ; ; 0494 ; ; 0495 ; ; 0496 READER: ; Reader input ; inputs the next data character fr							
0481;0482F8BC 3A0300LDAIOBYTE0483F8BF OFRRC0484F8C0 OFRRC0484F8C0 OFRRC0485F8C2 CDDDF8CALL0486F8C2 CDDDF8CALL0487; These routines return directly0488; to PUNCH's caller.0489F8C5 38F9DW0490F8C7 44F9DW0491F8C9 44F9DW0492F8CB 3EF9DW0494;0495;0496READER:; Reader input0497; inputs the next data character from the					; to th	e approp	riate device according to bits
0482F8BC 3A0300LDAIOBYTE;Get I/O redirection byte0483F8BF OFRRC;Move bits 5,4 to 2,10484F8C0 OFRRC0485F8C1 OFRRC0486F8C2 CDDDF8CALL0487;These routines return directly0488;These routines return directly0489F8C5 38F9DW0490F8C7 4AF9DW0491F8C9 44F9DW0493;0493;0494;0495;0496READER:; inputs the next data character from the						f IOBYTE	
0483F8BF OFRC;Move bits 5,4 to 2,10484F8C0 OFRRC0485F8C1 OFRRC0486F8C2 CDDDF8CALL0487;Select correct PUNCH routine0488;These routines return directly0489;to PUNCH's caller.0489F8C5 38F9DW0490F8C7 4AF9DW0491F8C9 44F9DW0492F8CB 3EF9DW0493;0494;0495;0496READER:; inputs the next data character from the	0482		0300				
0485 F8C1 0F RRC 0486 F8C2 CDDDF8 CALL Select\$Routine\$21 ;Select correct PUNCH routine 0487 ;These routines return directly ;to PUNCH's caller. 0488 ;to PUNCH's caller. 0489 F8C5 38F9 DW 0490 F8C7 4AF9 DW 0491 F8C9 44F9 DW 0492 F8CB 3EF9 DW 0493 ; 0494 ; 0495 ; 0496 READER: ;Reader input 0497 ; inputs the next data character from the							
0486 F8C2 CDDDF8 CALL Select\$Routine\$21 ;Select correct PUNCH routine ;These routines return directly 0487 ;These routines return directly ;These routines return directly 0488 ;These routines return directly 0489 F8C5 38F9 DW 0490 F8C7 4AF9 DW 0491 F8C9 44F9 DW 0492 F8CB 3EF9 DW 0493 ; 0494 ; 0495 ; 0496 READER: ;Entered directly from BIOS JMP vector; 0497 ; inputs the next data character from the							
0488 ; to PUNCH's caller. 0489 F8C5 38F9 DW Teletype\$Output ;00 <- IOBYTE bits 1,0	0486		DDF8		Select\$Routine\$	21	
0489F8C538F9DWTeletype\$Output;00 <- IOBYTE bits 1,00490F8C74AF9DWDummy\$Output;010491F8C944F9DWCommunication\$Output;100492F8C83EF9DWTerminal\$Output;110493;							
0490 F8C7 AAF9 DW Dummy\$Output ;01 0491 F8C9 44F9 DW Communication\$Output ;10 0492 F8CB 3EF9 DW Terminal\$Output ;11 0493 ;	0489						
0492 F8CB 3EF9 DW Terminal\$Output ;11 0493 ; 0494 ; 0495 ; 0496 READER: ;Reader input 0497 ;Entered directly from BIOS JMP vector; 0498 ; inputs the next data character from the							;01
0493 ; 0494 ; 0495 ; 0496 READER: ;Reader input 0497 ;Entered directly from BIOS JMP vector; 0498 ; inputs the next data character from the		F8CB 3E	F9				
0495 ; 0496 READER: ;Reader input 0497 ;Entered directly from BIOS JMP vector; 0498 ; inputs the next data character from the	0493			•			·
0496 READER: ;Reader input 0497 ;Entered directly from BIOS JMP vector; 0498 ; inputs the next data character from the				;			
0497 ;Entered directly from BIOS JMP vector; 0498 ; inputs the next data character from the				, READER:	;Reader	input	
	0497				;Entere	d direct	
···· , react device fillo the Hilegister							
					, , cabe		

0500				:The ap	propriat	e device	is sele	cted acc	ording	
0501						f IOBYTE			_	
0502				;						
0503		3A0300	LDA	IOBYTE			0 redire		te	
0504 0505	F8D0 F8D1		RRC. CALL	Select\$Routine\$	21.		its 3,2		routine	
0506	FODI	CDDDFO	CALL	Selectanouthea	21	;Select correct READER routine ;These routines return directly				
0507						; to READER's caller.				
0508	F8D4		DW	Teletype\$Output		;00 <- IOBYTE bits 1,0				
0509	F8D6		DW	Dummy\$Output		;01				
0510	F8D8		DW DW	Communication\$0		;10				
0511 0512	F8DA	JEF 7	DW	Terminal\$Output		;11				
0513			;							
0514			;							
0515			;							
0516			Select\$Rou	tine:					ed address	
0517 0518						alue of			according to	
0519	F8DC	07	RLC			select v			2.1	
0520		••				der to d				
0521			;							
0522			Select\$Rou	tine\$21:					selection bits	
0523 0524	F8DD	F606	ANI	0000\$0110B		lready i e just b		, 1		
0525	F8DF		XTHL	0000#01100		first wo		dresses	after	
0526					; CALL	instruct	ion			
0527	F8E0		MOV	E,A		selecti	on value	to addr	ess table	
0528	F8E1		MVI	D, O	; base					
0529 0530	F8E3	19	DAD	D		selected utine ad				
0531	F8E4	7F	MOV	A, M	;LS byt		uress In			
0532	F8E5		INX	н	;HL ->	MS byte				
0533	F8E6		MOV	Н,М	;MS byt	e				
0534	F8E7		MOV	L,A		routine				
0535	F8E8		XTHL			stack -				
0536 0537	F8E9	69	RET		; iranst	er to se	lected ro	Jutine		
0538			;							
0539			;							
0540			; Input/O	utput Equates						
0541	OOED	-	; Teletype\$S	+ = + us # Post		EQU	OEDH			
0543	OOEC		Teletype\$B		EQU	OECH	0CDH			
0544	0001			utput\$Ready		EQU	0000\$000	D1B	;Status mask	
0545	0002		Teletype\$I	nput\$Ready		EQU	0000\$00	IOB	;Status mask	
0546			1							
0547 0548	0001 0002		Terminal\$S Terminal\$D		EQU	EQU 02H	01H			
0548	0002			utput\$Ready	EQU	EQU	0000\$000	1B	;Status mask	
0550	0002		Terminal\$I			EQU	0000\$00:		;Status mask	
0551			;							
0552	OOED	=		ion\$Status\$Port	EQU	OEDH	0501			
0553	00EC 0001			ion\$Data\$Port ion\$Output\$Ready	FOU	EQU 0000\$00	OECH	;Status	mack	
0554	0001			ion\$Input\$Ready	EQU	0000\$00		;Status		
0556			;					,		
0557	OODF			ion\$Baud\$Mode		EQU	ODFH		;Mode Select	
0558	OODE	=	Communicat	ion\$Baud\$Rate		EQU	ODEH		;Rate Select	
0559			2							
0560			; ; Serial ·	device control t	ables					
0562			; Serial ;							
0563			; In orde	r to reduce the						
0564				e low-level driv					orts.	
0565				y to the low—lev iate control tab		r, HL PO	ints to 1	une.		
0566			; appropr	rate control (ab						
0568			, Teletype\$T	able:						
0569	F8EA		DB	Teletype\$Status						
0570	F8EB		DB	Teletype\$Data\$P						
0571	F8EC F8ED		DB DB	Teletype\$Output Teletype\$Input\$						
0572	r 860	02	, ,	ists()bs±10bd(»	nedu y					
0574			, Terminal\$T	able:						
0575	F8EE	01	DB	Terminal\$Status	\$Port					
L										

Figure 6-4. (Continued)

0576	F8EF 02	DB	Terminal\$ Data\$Port						
0577	F8F0 01	DB	Terminal\$Output\$Ready						
0578	F8F1 02	DB	Terminal\$Input\$Ready						
0579									
0580		Communicat	tion\$Table:						
0581	F8F2 ED	DB	Communication\$Status\$P	or*					
0582	F8F3 EC	DB	Communication\$Data\$Port						
0583	F8F4 01	DB	Communication\$Output\$R						
0584	F8F5 02	DB	Communication\$Input\$Re	ady					
0585		;							
0586		;							
0587									
0588		;							
0589			llowing routines are "ca	llad" by Calaat#Pouting					
0590		, ine io.	form the low-level input	fied by SelectProutine					
		; to peri	form the low-level input	Joutput					
0591		<u>.</u>							
0592			In\$Status:						
0593	F8F6 21EAF8	LXI	H,Teletype\$Table	;HL -> control table					
0594	F8F9 C34BF9	JMP	Input\$Status	;Note use of JMP. Input\$Status					
0595				; will execute the RETurn.					
0596		;		,					
0597			In\$Status:	•					
0598	F8FC 21EEF8		H, Terminal \$Table	;HL -> control table					
		LXI							
0599	F8FF C34BF9	JMP	Input\$Status	Note use of JMP. Input\$Status					
0600				; will execute the RETurn.					
0601		;							
0602		Communicat	tion\$In\$Status;						
0603	F902 21F2F8	LXI	H.Communication\$Table	:HL -> control table					
0604	F905 C34BF9	JMP	Input\$Status	Note use of JMP. Input\$Status					
0605		0.11	Impattotatas	; will execute the RETurn.					
				, will execute the Acidin.					
0606		, , ,		B					
0607		Dummy\$In\$;Dummy status, always returns					
0608	F908 3EFF	MVI	A, OFFH	; indicating incoming data is ready					
0609	F90A C9	RET							
0610		;							
0611		;							
0612			Dut\$Status:						
0613	F90B 21EAF8	LXI	H, Teletype\$Table	;HL -> control table					
	F90E C356F9	JMP		;Note use of JMP. Output\$Status					
0614	FYUE 0356F9	JMP	Output\$Status						
0615				; will execute the RETurn.					
0616		;							
0617		Terminal\$(Dut\$Status:						
0618	F911 21EEF8	LXI	H,Terminal\$Table	;HL -> control table					
0619	F914 C356F9	JMP	Output\$Status	Note use of JMP. Output\$Status					
0620				; will execute the RETurn.					
0621				, will execute the herdini					
		, Commundant	tion\$Out\$Status:						
0622									
0623	F917 21F2F8	LXI	H,Communication\$Table	;HL -> control table					
0624	F91A C356F9	JMP	Output\$Status	;Note use of JMP. Output\$Status					
0625				; will execute the RETurn.					
0626		;							
0627		Dummy\$Out\$	Status:	;Dummy status, always returns					
0628	F91D 3EFF	MVI	A.OFFH	; indicating ready for output					
0629	F91F C9	RET		, including ready for eacher					
0630	1 2 11 6 2	:							
		•							
0631		1							
0632		Teletype\$							
0633	F920 21EAF8	LXI	H,Teletype\$Table	;HL -> control table					
0634	F923 C360F9	JMP	Input\$Data	;Note use of JMP. Input\$Data					
0635				; will execute the RETurn.					
0636		;							
0637		Terminal\$	Incut:						
0638	F926 21EEF8	LXI	H,Terminal\$Table	;HL -> control table					
0639	Licelo			; will execute the RETurn.					
0640	F929 CD60F9	CALL	Input\$Data	; will execute the REfurn. ;** Special case **					
	F727 CUOURY	CALL	Inputabala						
0641		···-		;Input\$Data will return here					
0642	F92C E67F	ANI	7FH	; so that parity bit can be set O					
0643	F92E C9	RET							
0644		;							
0645		Communicat	tion\$Input:						
0646	F92F 21F2F8	LXI	H,Communication\$Table	;HL -> control table					
0647	F932 C360F9	JMP	Input\$Data	Note use of JMP. Input\$Data					
0648		0.1		; will execute the RETurn.					
				, will execute the neturn.					
0649		1							
0650		Dummy\$Inp		;Dummy input, always returns					
0651	F935 3E1A	MVI	A,1AH	; indicating CP/M end of file					

0652	F937	C9	RET			
0653			;			
0654 0655			;			
0656			;			
0657			Teletype\$C	Output:		
0658		21EAF8		H,Teletype\$Tabl	e	;HL -> control table
0659	F93B	C370F9	JMP	Output\$Data		Note use of JMP. Output\$Data
0660 0661			_			; will execute the RETurn.
0662			; Terminal\$C	utout.		
0663	F93E	21EEF8		H,Terminal\$Tabl	e	;HL -> control table
0664					-	; will execute the RETurn.
0665	F941	C370F9	JMP	Output\$Data		Note use of JMP. Output\$Data
0666						; will execute the RETurn.
0667			1			
0668 0669	E944	21F2F8	LXI	ion\$Output: H,Communication	#T-61-	;HL -> control table
0670		C370F9	JMP	Output\$Data	* aute	;Note use of JMP. Output\$Data
0671				Catpattoata		; will execute the RETurn.
0672			;			
0673			Dummy\$Outp	out:		;Dummy output, always discards
0674	F94A	C9	RET			; the output character
0675			;			
0677			;			
0678			;			
0679				ire the general p		
0680						priate control table.
0681			; For out	put, the C regis	ter cont	ains the data to be output.
0682 0683			; Input\$Stat		. Poturo	with $A = 00H$ if no incoming data,
0684			Input#Stat			vise A = nonzero.
	F94B	7E	MOV	A, M	;Get st	itus port
0686	F94C	3250F9 DB	STA	Input\$Status\$Po	rt	;*** Self-modifying code ***
0687	F94F	DB	DB	IN	;Input	o A from correct status port
0688						
0689	F950	00	Input\$Stat	us#Port:	;<- Set	above
0691	F951		INX	H		. to point to input data mask
0692	F952		INX	н	,	
0693	F953		INX	H Star		
0694	F954	A6	ANA	M	;Mask w	th input status
0695	F955	C9	RET			
0697			;			
0698			, Output\$Sta	tus:	Return	with $A = 00H$ if not ready for output
0699						vise A = nonzero.
0700	F956		MOV	Α,Μ		itus port
0701		325BF9	STA			;*** Self-modifying code ***
0702	F95A	DB	DB	IN	;Input	o A from correct status port
0703			; Output\$Sta	turtPorts		
0705	F95B	00	DB	00	;<- Set	above
0706	F95C	23	INX	н		. to point to output data mask
0707	F95D	23	INX	н		
0708	F95E	A6	ANA	M	;Mask w	th output status
0709	F95F	C9	RET			
0710			;			
0712			, Input\$Data		Return	with next data character in A.
0713				-		or status routine to indicate
0714					; incom:	ng data.
0715	F960		PUSH	н		ntrol table pointer
0716		CD4BF9	CALL	Input\$Status		out status in zero flag
0717	F964	E1 CA60F9	POP JZ	H Input\$Data		control table pointer
0719	F968		INX	Inputsuata H		Itil incoming data lata port
0720	F969		MOV	A. M	;Get da	a port
0721	F96A	326EF9	STA		;*** Se	f-modifying code ***
0722	F96D		DB	IN		o A from correct data port
0723			1			
0724	EQ1E	00	Input\$Data DB	\$Port: 0		above
0725	F96E F96F		RET	v	;<- Set	adove
0727	1.201		;			
			the second s			

Figure 6-4. (Continued)

0728					
0729			, Output\$Dat		;Output the data character in the C register.
0730			output#bat		;Wait for status routine to indicate device
0731					; ready to accept another character
0732) E5	PUSH	н	Save control table pointer
0733		CD56F9	CALL	 Output\$Status	Get output status in zero flag
0734			POP	н	Recover control table pointer
0735		CATOF9	JZ	Output\$Data	;Wait until ready for output
0736			INX	H	;HL -> output port
0737			MOV	A, M	;Get output port
0738	F97A	327FF9	STA	Output\$Data\$Por	
0739	F970	79	MOV	A,C	;Get data character to be output
	F97E	. D3	DB	OUT	;Output data to correct port
0741			;		,,
0742			Output\$Dat	a\$Port:	
0743	F97F	00	DB	0	;<- Set above
0744	F980	0 09	RET		
0745			;		
0746			;		
0747			; High le	evel diskette dri	vers
0748			;		,
0749			; These d	lrivers perform t	the following functions:
0750			;		
0751			; SELDSK	Select a specif	ied disk and return the address of
0752			;	the appropriate	e disk parameter header
0753			; SETTRK	Set the track r	number for the next read or write
0754			; SETSEC	Set the sector	number for the next read or write
0755			; SETDMA	Set the DMA (re	ad/write) address for the next read or write.
0756			; SECTRAN	l Translate a log	ical sector number into a physical
0757			; HOME	Set the track t	o 0 so that the next read or write will
0758			;	be on Track O	
0759			;		
0760					evel drivers are responsible for making
0761			; the 51	/4" floppy diske	ttes that use a 512-byte sector appear
0762			; to CP/M	l as though they	used a 128-byte sector. They do this
0763			; by usin	ng what is called	blocking/deblocking code,
0764					l later in this listing,
0765			; just pr	ior to the code	itself.
0766			;		
0767			;		
0768			;		
0769			; Disk pa	rameter tables	
0770			;		
0771					3, these describe the physical
0772			; charact	eristics of the	disk drivest In this example BIOS,
0773			; there a	re two types of	disk drives; standard single-sided,
0774					double-sided, double-density 5 1/4"
0775			; diskett	es.	
0776			;		
0777			; The sta	ndard 8" diskett	es do not need to use the blocking/
0778					e 5 1/4" drives do. Therefore an additional
0779			; byte ha	s been prefixed	to the disk parameter block to
0780					ach logical disk's physical
0781			; diskett	e type, and whet	her or not it needs deblocking.
0782			;		
0783			7		
0784				finition tables	
0785			;		
0786					arameter headers, with one entry
0787					, and disk parameter blocks, with
0788			; either	one parameter bl	ock per logical disk or the same
0789				er block for sev	eral logical disks.
0790			,		
0791			7 Dielebrar		Described in Observice O
0792				eter\$Headers:	;Described in Chapter 3
0793			;		1 Dick A. (5.1/4) Dickotto)
0795	E004	6BFB	DW	Floppy\$5\$Skewta	l Disk A: (5 1/4" Diskette) ble
0796		0000000000	DW	0,0,0 Directory#Buffe	Reserved for CP/M
0797		C1F9	DW	Directory\$Buffe	
0798		42FA	DW	Floppy\$5\$Parame	
		61FA		Disk\$A\$Workarea	
0800	F 78F	C1FA	DW	Disk\$A\$Allocati	on avector
0801			;		
0802	E004	ADED	DW		1 Disk B: (5 1/4" Diskette)
0803	F 771	6BFB	Liw	Floppy\$5\$Skewta	ble ;Shares same skew table as A:

Figure 6-4. (Continued)

0804 F993 0000000000 DW 0.0.0 ;Reserved for CP/M ;Share same buffer as A: 0805 F999 C1F9 DW Directory\$Buffer 0806 F99B 42FA DW Floppy\$5\$Parameter\$Block ;Same DPB as A: 0807 F99D 81FA DW Disk\$B\$Workarea ;Private work area Private allocation vector 0808 F99F D7FA nш Disk\$B\$Allocation\$Vector 0809 ; ;Logical Disk C: (8" Floppy) MaSkewtable ;8" skew table 0810 Floppy\$8\$Skewtable 0811 F9A1 B3FB DW E9A3 0000000000 nω Reserved for CP/M 0812 0,0,0 F9A9 C1F9 DW 0813 Directory\$Buffer :Share same buffer as A: 0814 F9AB 52FA DW Floppy\$8\$Parameter\$Block 0815 F9AD A1FA DW Disk\$C\$Workarea ;Private work area F9AF EDFA DW Disk\$C\$Allocation\$Vector ;Private allocation vector 0816 0817 ; 0818 ;Logical Disk D: (8" Floppy) 0819 F9B1 6BFB nw Floppy\$5\$Skewtable ;Shares same skew table as A: 0820 F9B3 0000000000 DW 0,0,0 Reserved for CP/M ;Share same buffer as A: ;Same DPB as C: 0821 F9B9 C1F9 DW Directory\$Buffer 0822 F9BB 52FA DW Floppy\$8\$Parameter\$Block 0823 F9BD B1FA F9BF OCFB DΜ Disk\$D\$Workarea ;Private work area
;Private allocation vector пы Disk\$D\$Allocation\$Vector 0824 0825 0826 0827 Directory\$Buffer: DS 0828 F9C1 128 0829 : 0830 0832 : 0833 Disk Types . 0834 Floppy\$5 ;5 1/4" mini floppy 0835 0001 = EQU 1 0836 0002 =Floppy\$8 EQU 2 ;8" floppy (SS SD) 0837 0838 Blocking/deblocking indicator 0839 0840 0080 =Need\$Deblocking FOIL 1000\$00008 :Sector size > 128 hytes 0841 0842 0843 Disk parameter blocks . 0844 : 5 1/4" mini floppy 0845 ; 0846 - 2 ;Extra byte prefixed to indicate 0847 0848 ; disk type and blocking required 0849 FA41 81 DB Floppy\$5 + Need\$Deblocking 0850 Floppy\$5\$Parameter\$Block: 0851 FA42 4800 nu 72 ;128-byte sectors per track ;Block shift 0852 FA44 04 DB Δ 15 ;Block mask 0853 FA45 OF **NR** FA46 01 FA47 AE00 FA49 7F00 ;Extent mask 0854 DB 1 174 :Maximum allocation block number 0855 ΠW 0856 DW 127 ;Number of directory entries - 1 0857 FA4B CO DB 1100\$0000B ;Bit map for reserving 1 alloc. block 0000\$0000B for file directory 0858 FA4C 00 DB ; 0859 FA4D 2000 ;Disk changed work area size DW 32 0860 FA4F 0100 DW ;Number of tracks before directory 0861 0862 ; Standard 8" Floppy 0863 ; 0864 ;Extra byte prefixed to DPB for 0865 ; this version of the BIOS FA51 02 ;Indicates disk type and the fact 0866 DB Floppy\$8 ; that no deblocking is required 0867 Floppy\$8\$Parameter\$Block: 0868 0869 FA52 1A00 26 Sectors per track DW FA54 03 DB 3 ;Block shift 0870 FA55 07 7 Block mask 0871 DB 0872 FA56 00 DB 0 ;Extent mask 0873 FA57 F200 DW 242 ;Maximum allocation block number 0874 FA59 3F00 DW 63 :Number of directory entries - 1 1100\$0000B 0875 FA5B CO **DB** ;Bit map for reserving 2 alloc. blocks ; for file directory 0876 FA5C 00 **DR** 0000\$0000B ;Disk changed work area size 0877 FA5D 1000 nw 16 :Number of tracks before directory 0878 FA5F 0200 nы 2 0879 ; 0880 .

Figure 6-4. (Continued)

0881				ork areas					
0882			;						
0883			; These a	are used b	ly the BL	JUS to de	etect any un	expected	
0885							ill automati -only status		
0886			; such a	changed d	IJAKELLE	to reau	onry status	•	
0887	FA61		Disk\$A\$Wor	'karea:	DS	32	; A:		
0888	FA81		Disk\$B\$Wor	'karea:	DS	32	; B:		
0889	FAA1		Disk\$C\$Wor	'karea:	DS	16	; C:		
0890	FAB1		Disk\$D\$Wor	'karea:	DS	16	; D:		
0891			,						
0892			; . Dick al	location	vantare				
0894			; Disk al	rocation	vectors				J
0895				are used b	v the BI	00S to ma	aintain a bi	t map of	
0896			; which a	allocation	blocks	are used	d and which	are free.	
0897							ation block		the
0898			; express	sion of th	e form (allocati	ion blocks/8	0+1.	
0899	FAC1		; Disk\$A\$A1]			DS	(174/8)+1		
0901	FAD7		Disk\$B\$A11			DS	(174/8)+1 (174/8)+1	; A: : B:	
0902	1 1107		:		ector	50	(1)4/8/41	, 5.	
0903	FAED		Disk\$C\$A11	location\$V	ector	DS	(242/8)+1	; C:	
0904	FBOC		Disk\$D\$Al]	location\$V	ector .	DS	(242/8)+1	; D:	
0905			;						
0906	0000	_	; 						
0907	0004	-	Number\$of\$	Progical»	915KS		EQU 4		
0909			, :						
0910			SELDSK:			:Select	disk in C		
0911						;C = 0 1	or drive A,	1 for B,	etc.
0912						;Return	the address	of the a	ppropriate
0913							arameter he		
0914 0915						; 17 the	e selected d	isk does	not exist.
0916	FB2B	210000	LXI	н, о		7	an error		
0917	FB2E	79	MOV	A,C			if requested	disk val	id
0918	FB2F	FE04	CPI	Number \$c	f\$Logica	al\$Disks			
0919	FB31	DO	RNC			;Return	if > maximu	m number (of disks
0920	EB33	32EAFB	, STA	Selected	d Dial.				
0922	FD32	JZEAFD	314	Selected	I DI SK		elected disk to return D		c .
0923	FB35	6F	MOV	L,A			isk into wor		-
0924	FB36	2600	MVI	н, о					
0925							e offset dow		
0926							table by m		
0928	FB38	29	DAD	н		; parame ; *2	eter header	length (1	6 Bytes)
0929	FB39		DAD	н		; *4			
0930	FB3A		DAD	н		; *8			
0931	FB3B		DAD	н		; *16			
0932		1181F9	LXI	D,Disk\$P	'ar ameter	\$Headers	;Ge	t base ad	dress
0933	FB3F		DAD	D		;DE -> 4	Appropriate	DPH	
0934	FB40	ED	PUSH	н		;save DF	°H address		
0936			,			Access	disk parame	ter block	
0937							ract specia		
0938							fies disk t		
0939						; debloc	king is req	uired	
0940				B 4.4		;			
0941	FB41	110A00		D, 10 D			3 pointer of)PB address		PH
0943	FB45		MOV	É,M			3 address in		
0944	FB46		INX	H		,			
0945	FB47		MOV	D,M					
0946	FB48		XCHG			;DE -> D			
0947 0948	FB49 FB4A		DCX MOV	Н А, М			refix byte		
0948	FB4A FB4B		ANI	A,M OFH			fix byte disk type		
0950		32FAFB	STA	Disk\$Typ	e		r use in lo	w-level d	river
0951	FB50		MOV	A, M			ther copy o		
0952	FB51	E680	ANI	Need\$Deb	locking		;Isolate de		
0953		32F9FB	STA	Deblocki	ng\$Requi				-level driver
0954	FB56		POP	н		;Recover	DPH pointe	r	
0955	FB57	67	RET						
			*						

0957 0958 Set logical track for next read or write 0959 SETTRK 0960 0961 FB58 60 ;Selected track in BC on entry MOV H,B 0962 FB59 69 MOV L.C 0963 FB5A 22EBFB SHLD Selected\$Track ;Save for low-level driver FB5D C9 0964 RET 0965 : 0966 ; 0967 Set logical sector for next read or write : 0968 ; 0969 : 0970 SETSEC: ;Logical sector in C on entry 0971 FB5E 79 MOV A.C 0972 FB5F 32EDFB STA Selected\$Sector ;Save for low-level driver 0973 FB62 C9 RET 0974 0975 ; ; 0976 Set disk DMA (input/output) address for next read or write : 0977 0978 FB63 0000 DMA\$Address: DW 0 :DMA address 0979 0980 SETDMA: ;Address in BC on entry 0981 FB65 69 MOV L,C ;Move to HL to save 0982 FB66 60 MOV Н, В 0983 FB67 2263FB SHLD DMA\$Address ;Save for low-level driver 0984 FB6A C9 RET 0985 : 0986 ; 0987 Translate logical sector number to physical ; 0988 : 0989 : Sector translation tables 0990 These tables are indexed using the logical sector number, ; 0991 and contain the corresponding physical sector number. : 0992 0993 Floppy\$5\$Skewtable: ;Each physical sector contains four 0994 ; 128-byte sectors. 0995 Physical 128b Logical 128b Physical 512-byte 0996 FB6B 00010203 DB 00,01,02,03 ;00,01,02,03 0) 4 0997 FB6F 10111213 DB 16,17,18,19 ;04,05,06,07) 0998 FB73 20212223 **DR** 32, 33, 34, 35 ;08,09,10,11 R) 12, 13, 14, 15 28, 29, 30, 31 0999 ;12,13,14,15 FB77 OCODOFOF DB 3 7) Head 1000 ;16,17,18,19 ō FB7B 1C1D1E1F DB) 1001 FB7F 08090A0B DB 08,09,10,11 :20.21.22.23 2) DB 24,25,26,27 1002 FB83 18191A1B ;24,25,26,27 6 04,05,06,07 ;28,29,30,31 1003 FB87 04050607 DB 1) 1004 FB8B 14151617 DB 20,21,22,23 ;32,33,34,35 5) 1005 ; 1 1006 FB8F 24252627 DB 36,37,38,39 ;36,37,38,39 0 FB93 34353637 52, 53, 54, 55 ;40,41,42,43 1007 DB 4 3 1008 FB97 44454647 DB 68,69,70,71 ;44,45,46,47 8 3 1009 FB9B 30313233 DB 48,49,50,51 ;48,49,50,51 з] Head ; 52, 53, 54, 55 1010 FB9F 40414243 DB 64,65,66,67 7 1 1 1011 FBA3 2C2D2E2F DB 44, 45, 46, 47 ;56,57,58,59 2 3 30303F3F ;60,61,62,63 1012 FRA7 **DB** 60,61,62,63 6 1 ;64,65,66,67;68,69,70,71 FBAB 28292A2B DB 40.41.42.43 3 1013 1 FBAF 38393A3B 56, 57, 58, 59 3 1014 **DB** 5 1015 1016 Floppy\$8\$Skewtable: 1017 ;Standard 8" Driver 01,02,03,04,05,06,07,08,09,10 Logical sectors 1018 1019 FBB3 01070D1319 ĎВ 01,07,13,19,25,05,11,17,23,03 Physical sectors 1020 ; 1021 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 Logical sectors 1022 FBBD 090F150208 **n**R 09, 15, 21, 02, 08, 14, 20, 26, 06, 12 ;Physical sectors 1023 : 21,22,23,24,25,26 18,24,04,10,16,22 1024 Logical sectors FBC7 1218040A10 **NR** 1025 Physical sectors 1026 1027 SECTRAN: 1028 ;Translate logical sector into physical ;On entry, BC = logical sector number ; DE -> appropriate skew table 1029 1030 1031 1032 ;on exit, HL = physical sector number

Figure 6-4. (Continued)

1033 1034 1035 1036 1037 1038	FBCD FBCE FBCF FBD0 FBD2	09 6E 2600	XCHG DAD MOV MVI RET ;	B L,M H,O	;Add or ;Get pl	nysical s	le base sector number ector number -bit value	
1039 1040 1041 1042 1043 1044 1045 1046			; ; HOME:		;Beford ; if the ; that ; a fle	e doing t ne physic must be	ted logical disk to track 0. his, a check must be made to see al disk buffer has information written out. This is indicated by Write\$Buffer, set in the de.	
1047 1048 1049 1050	FBD6	3AE9FB B7 C2DDFB	LDA ORA JNZ	Must\$Write\$Bu A HOME\$No\$Write			if physical buffer must ritten out to disk	
1051 1052		32E8FB	STA	Data\$In\$Disk\$			o indicate that buffer low unoccupied.	
1053 1054 1055 1056 1057		0E00 CD58FB C9	HOME\$No\$Wr MVI CALL RET	C,O SETTRK			o track O (logically lotual disk operation occurs)	
1058 1059 1060 1061 1062			; ; Bata written to or read from the mini-floppy drive is transferred ; via a physical buffer that is actually 512 bytes long (it was ; declared at the front of the BIOS and holds the "one-time" ; initialization code used for the cold boot procedure).					
1063 1064 1065 1066 1067 1068 1069 1070			; of actu ; current ; a 128-b	al disk I/O by ly residing in	storing the Phys or" that a	the disk, ical Buff	o minimize the amount track, and physical sector er. If a read request is for s in the physical buffer,	
1071 1072 1073 1074 1075 1076 1077	0800 0012 0004 0048 0003 0002	= = =	Physical\$S CPM\$Sec\$Pe CPM\$Sec\$Pe Sector\$Mas Sector\$Bit	r\$Track k	EQU EQU EQU EQU EQU EQU	CPM\$Sec	ll\$Sector\$Size/128 \$Per\$Physical*Physical\$Sec\$Per\$Track \$Per\$Physical−1 ;LOG2(CPM\$Sec\$Per\$Physical)	
1078 1079 1080 1081 1082 1083 1084 1085 1086 1087				; wh ;The ; BD ; bl. ; 12: ; th. ;The ; th	en it cal: allocated OS is set ock (it o 8-byte sec at has all	ls the WF /unalloca to write nly indic tor writ ready bee indicate	anded over by the BDOS ITE operation. Ited indicates whether the to an unallocated allocation ates this for the first e) or to an allocation block n allocated to a file. s if it is set to write to	
1087	0000		Write\$Allo Write\$Dire		EQU EQU	0 1		
1090 1091	0002	=	Write\$Unal ;	located	EQU	2		
1092 1093 1094	FBE3	00	Write\$Type ;	1	DB	0	;Contains the type of write ; indicated by the BDOS.	
1095 1096 1097			; In\$Buffer\$	Dk\$Trk\$Sec:			;Variables for physical sector ; currently in Disk\$Buffer in memory	
1097	FBE4	00	In\$Buffer\$	Disk:	DB	0	; These are moved and compared	
1099	FBE5		In\$Buffer\$		DW	ŏ	; as a group, so do not alter	
1100 1101	FBE7		In\$Buffer\$		DB	Ō	; these lines.	
1102 1103	FBE8		Data\$In\$Di		DB	0	;When nonzero, the disk buffer has ; data from the disk in it.	
1104 1105 1106	FBE9	00	Must\$Write	≸Buffer:	DB	0	;Nonzero when data has been ; written into Disk\$Buffer but ; not yet written out to disk	
1107 1108			; Selected\$D	k\$Trk\$Sec:	;Variat	oles for	selected disk, track, and sector	

1109						
11109	FBEA 00	Selected	Dicks	; (Se DB	lected b	y SELDSK, SETTRK,and SETSEC) ; These are moved and
1111	FBEB 0000	Selected		DW	ŏ	; compared as a group so
1112	FBED 00	Selected		DB	ŏ	; do not alter order.
1113					-	,
1114	FBEE 00	Selected	Physical\$Sector:	DB	0	;Selected physical sector derived
1115						; from selected (CP/M) sector by
1116						; shifting it right the number of
1117						; of bits specified by
1118						; Sector\$Bit\$Shift
1119		;				
1120	FBEF 00	Selected\$	Disk\$Type:	DB	0	;Set by SELDSK to indicate either
1121						; 8" or 5 1/4" floppy
1122	FBFO OO	Selected\$	Disk\$Deblock:	DB	0	;Set by SELDSK to indicate whether
1123 1124						; deblocking is required.
1124						
1126		Unallocat	ed\$Dk\$Trk\$Sec:			
1127		onariocat	edankali kaseci		;rara	neters for writing to a previously ; unallocated allocation block.
1128	FBF1 00	Unallocat	ad & Dieks	DB	0	; Unallocated allocation block. ; These are moved and compared
1129	FBF2 0000		ed\$Track:	DW	ŏ	; mese are moved and compared ; as a group so do not alter
1130	FBF4 00		ed\$Sector:	DB	ŏ	; as a group so not after ; these lines.
1131					•	
1132	FBF5 00	Unallocat	ed\$Record\$Count:	DB	0	Number of unallocated "records"
1133						; in current previously unallocated
1134						; allocation block.
1135						
1136	FBF6 00	Disk\$Errc	r\$Flag:	DB	0	Nonzero to indicate an error
1137						; that could not be recovered
1138						; by the disk drivers. BDOS will
1139						; output a "bad sector" message.
1140 1141		;				
1141		;riags us	ed inside the deb	0100610	g code	
	FBF7 00	MustePrav	ead\$Sector:	DB	0	Nonzero if a physical sector must
1144	1017 00	nust#i i ei	eau#Sector.	00	0	; be read into the disk buffer
1145						; either before a write to an
1146						; allocated block can occur, or
1147						; for a normal CP/M 128-byte
						; sector read
1148						
1148 1149	FBF8 00	Read\$Oper	ation:	DB	0	Nonzero when a CP/M 128-byte
1149 1150		Read\$0per	ation:	DB	0	
1149 1150 1151	FBF8 00 FBF9 00		ation: g\$Required:	DB DB	0 0	;Nonzero when a CP/M 128-byte
1149 1150 1151 1152	FBF9 00	Deblockin	g\$Required:	DB	0	;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK)
1149 1150 1151 1152 1153			g\$Required:		•	;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy
1149 1150 1151 1152 1153 1154	FBF9 00	Deblockin Disk‡Type	g\$Required:	DB	0	;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK)
1149 1150 1151 1152 1153 1154 1155	FBF9 00	Deblockin Disk‡Type ;	g\$Required:	DB	0	;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy
1149 1150 1151 1152 1153 1154 1155 1156	FBF9 00	Deblockin Disk‡Type ; ;	g\$Required: :	DB DB	0	;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8° or 5 1/4° floppy ; selected (set in SELDSK).
1149 1150 1151 1152 1153 1154 1155 1156 1157	FBF9 00	Deblockin Disk‡Type ; ; ; Read i	g\$Required: : n the 128-byte CF	DB DB 2/M sec	0 0 tor speci	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158	FBF9 00	Deblockin Disk‡Type ; ; ; Read i ; to sel	g\$Required: : n the 128-byte CF ect disk and to s	DB DB 2/M sec set tra	0 0 tor speci ck and se	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls sector. The sector will be read</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159	FBF9 00	Deblockin Disk‡Type ; ; ; Read i ; to sel ; into t	g\$Required: : n the 128-byte CF ect disk and to s	DB DB 2/M sec set tra	0 0 tor speci ck and se	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160	FBF9 00	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ;	g\$Required: : n the 128-byte CF ect disk and to s he address specif	DB DB 2/M sec set tra ied in	0 0 tor spec: ck and se the prev	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls zctor. The sector will be read vious call to set DMA address.</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159	FBF9 00	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; ; If rea	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk	DB DB 2/M sec set tra- ied in drive	0 0 tor spect ck and se the prev using sec	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls sector. The sector will be read</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161	FBF9 00	Deblockin Disk\$Type ; ; Read i ; to sel ; into t ; ; If rea ; debloc	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be	DB DB 2/M sec set tra- ied in drive	0 0 tor spect ck and se the prev using sec	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls zector. The sector will be read vious call to set DMA address. etors larger than 128 bytes,</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164	FBF9 00 FBFA 00	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ;	g\$Required: : ect disk and to s he address specif ding from a disk king code will be ysical sector.	DB DB C/M sec set tra ied in drive sused	0 0 tor speci ck and se the prev using sec to "unpac	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls cotor. The sector will be read vious call to set DMA address. ctors larger than 128 bytes, ck" a 128-byte sector from</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163	FBF9 00 FBFA 00 FBFB 3AF9FB	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; dfebloc ; the ph READ: LDA	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be	DB DB C/M sec set tra ied in drive sused	0 0 tor speci ck and se the prev using sec to "unpac	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls zector. The sector will be read vious call to set DMA address. etors larger than 128 bytes,</pre>
1149 1150 1151 1152 1153 1154 1155 1155 1155 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166	FBF9 00 FBFA 00 FBFB 3AF9FB FBFE B7	Deblockin Disk\$Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA	g\$Required: : ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A	DB DB 2/M sec set tra- ied in drive sused	0 0 tor spec: ck and se the prev using sec to "unpac ;Check ;(flag	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. ctors larger than 128 bytes, ck" a 128-byte sector from (if deblocking needed ; was set in SELDSK call)</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1161 1163 1164 1165 1165	FBF9 00 FBFA 00 FBFB 3AF9FB	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; dfebloc ; the ph READ: LDA	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ	DB DB 2/M sec set tra- ied in drive sused	0 0 tor spec: ck and se the prev using sec to "unpac ;Check ;(flag	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls sector. The sector will be read vious call to set DMA address. stors larger than 128 bytes, st" a 128-byte sector from < if deblocking needed</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1157 1158 1160 1161 1162 1163 1164 1165 1166	FBF9 00 FBFA 00 FBFB 3AF9FB FBFE B7	Deblockin Disk\$Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock	DB DB P/M sec set tra- ied in drive sused	0 0 tor speci tk and se the prev using sec to "unpac ;Check ;(flac ;No, u	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls zetor. The sector will be read vious call to set DMA address. ctors larger than 128 bytes, ctr a 128-byte sector from (if deblocking needed g was set in SELDSK call) use normal nondeblocked</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168	FBF9 00 FBFA 00 FBFB 3AF9FB FBFE B7	Deblockin Disk\$Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA	g\$Required: : ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ;The de	DB DB DB contra ied in drive e used uired contra co	0 0 tor spec; ck and se the prev using sec to "unpac ;Check ;Check ;flac ;No, t ng algor;	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls cetor. The sector will be read vious call to set DMA address. ctors larger than 128 bytes, ck" a 128-byte sector from (if deblocking needed g was set in SELDSK call) use normal nondeblocked ithm used is such</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1161 1162 1163 1164 1165 1166 1167 1168 1169	FBF9 00 FBFA 00 FBFB 3AF9FB FBFE B7	Deblockin Disk\$Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ;The de ; that	DB DB DB P/M sec set tra ied in drive used hired seblocki a rea	0 0 0 tor spec: ck and se the prev scheck ;(flag ;(flag ;(flag doperation)	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. etors larger than 128 bytes, ck" a 128-byte sector from (if deblocking needed g was set in SELDSK call) ise normal nondeblocked ithm used is such ion can be viewed</pre>
1149 1150 1151 1152 1153 1155 1156 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1169 1169 1170	FBF9 00 FBFA 00 FBFB 3AF9FB FBFE B7	Deblockin Disk\$Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; The de ; that ; up t	DB DB DB with trained inted in drived used inted tolockia inarea antil t	0 0 0 tor spec: ck and se the prev using se to "unpac ;(flag ;No, u ng algori d operati d operati	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. ctors larger than 128 bytes, ctors larger than 128 bytes, ctors</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1157 1158 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171	FBF9 00 FBFA 00 FBFB 3AF9FB FBFE B7	Deblockin Disk\$Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; The de ; that ; up c ; tho	DB DB DB v/M sec set tra- ied in drive sused uired cocki a rea until ti gh iti	0 0 0 ck and set the prev using set to "unpac ;Check ;No, to ;No, to algor d operati he actual was the i	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. etors larger than 128 bytes, etw a 128-byte sector from (if deblocking needed g was set in SELDSK call) use normal nondeblocked ithm used is such ion can be viewed i data transfer as first write to an</pre>
1149 1150 1151 1152 1153 1154 1155 1155 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1160 1171 1172	FBF9 00 FBFA 00 FBFA 3AF9FB FBFE 87 FBFF CA52FD	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ	g\$Required: : n the 128-byte Cf ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; The de ; that ; up u ; thou ; unal	DB DB DB v/M sec set tra- ied in drive sused uired cocki a rea until ti gh iti	0 0 tor spec: ck and set the prev using set to "unpac ;Check ;(flag ;No, u ng algori d operati he actual was the fd allocal d allocal	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. ettors larger than 128 bytes, cstr a 128-byte sector from (if deblocking needed g was set in SELDSK call) use normal nondeblocked ithm used is such ithm used is such ithm used is such ithm used is such ithm used is such itat transfer as first write to an tion block.</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1157 1158 1160 1161 1161 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173	FBF9 00 FBFA 00 FBFA 3AF9FB FBFE B7 FBFF CA52FD FC02 AF	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; The de ; that ; un d A	DB DB DB C/M sec set tra- ied in drive of used uired c c c c c c c c c c c c c c c c c c c	0 0 0 tor spec; ck and se the prev using sec to "unpac ;(flag ;No, u ng algor; d operati he actual was the f d allocat y Set i	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls zector. The sector will be read vious call to set DMA address. tors larger than 128 bytes, ck" a 128-byte sector from (if deblocking needed g was set in SELDSK call) use normal nondeblocked ithm used is such ion can be viewed i data transfer as first write to an tion block. the record count to 0</pre>
1149 1150 1151 1152 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174	FBF9 00 FBFA 00 FBFA 00 FBFB 3AF9FB FBFE 87 FBFF CA52FD FC02 AF FC02 AF FC03 32F5FB	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ XRA STA	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; The de ; that ; un d A	DB DB DB C/M sec set tra- ied in drive of used uired c c c c c c c c c c c c c c c c c c c	0 0 tor spec: ck and set the prev scheck prev ;Check ;(flac ;No, u ng algorid d operati he actual was the id d allocat ;Stet i unt; f fd	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. etors larger than 128 bytes, ck" a 128-byte sector from (if deblocking needed g was set in SELDSK call) use normal nondeblocked ithm used is such ion can be viewed L data transfer as first write to an tion block. the record count to 0 or first "write"</pre>
1149 1150 1151 1152 1153 1154 1156 1156 1157 1158 1159 1160 1161 1162 1164 1165 1166 1167 1168 1169 1170 1170 1171 1172 1174	FBF9 00 FBFA 00 FBFA 00 FBFB 3AF9FB FBFE B7 FBFF CA52FD FC02 AF FC03 32F5FB FC06 3C	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ XRA STA INR	g\$Required: : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; The de ; that ; un d A	DB DB DB C/M sec set tra- ied in drive of used uired c c c c c c c c c c c c c c c c c c c	0 0 tor spec: ck and set the prev using set to "unpac ;(flac ;(flac ;(flac ;(flac ;(No, u ng algori d operati d operati set i ;Set i unt ; fc ;Indic	<pre>shonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls cetor. The sector will be read vious call to set DMA address. ctors larger than 128 bytes, ctors larger than 128 bytes, ctors</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1175 1175	FBF9 00 FBFA 00 FBFA 00 FBFB 3AF9FB FBFE B7 FBFF CA52FD FC03 32F5FB FC03 32F5FB FC07 32F8FB	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ XRA STA STA	g\$Required: : : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; The de ; that ; up to ; that y un do ; unal A Unallocated\$Rec A Read\$Operation	DB DB DB V/M sec set tra- ied in drive seused hired contection locate cord\$Co	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	<pre>shonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. ectors larger than 128 bytes, ettors larger than 128 bytes, ettor</pre>
$\begin{array}{c} 1149\\ 1150\\ 1151\\ 1152\\ 1153\\ 1154\\ 1155\\ 1157\\ 1158\\ 1157\\ 1158\\ 1160\\ 1161\\ 1162\\ 1163\\ 1163\\ 1163\\ 1168\\ 1167\\ 1168\\ 1167\\ 1171\\ 1171\\ 1173\\ 1174\\ 1177\\ 1178\\ 1176\\ 1177\\ 1178\\ \end{array}$	FBF9 00 FBFA 00 FBFA 00 FBFB 3AF9FB FBFE B7 FBFF CA52FD FC02 AF FC03 32F5FB FC06 3C	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ XRA STA INR	g\$Required: : : ect disk and to she address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ;The de ; that ; up u ; thou ; unal A Unallocated\$Rec	DB DB DB V/M sec set tra- ied in drive seused hired contection locate cord\$Co	0 0 tor spec: ck and set the prev using set to "unpad operation algorid doperation set to allocal station for ; Indic ; Indic ; that a to set to allocal set to set to set to set to set to set to set to set to set to	<pre>shonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. etors larger than 128 bytes, sk" a 128-byte sector from (if deblocking needed g was set in SELDSK call) ise normal nondeblocked lithm used is such toon can be viewed l data transfer as first write to an tion block. the record count to 0 or first "write" rate that it is really a read at is to be performed i force a preread of the sector</pre>
1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1160 1162 1163 1164 1165 1166 1167 1168 1169 1170 1177 1173	FBF9 00 FBFA 00 FBFA 00 FBFB 3AF9FB FBFE B7 FBFF CA52FD FC03 32F5FB FC03 32F5FB FC07 32F8FB	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ XRA STA STA	g\$Required: : : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; The de ; that ; up to ; that y un do ; unal A Unallocated\$Rec A Read\$Operation	DB DB DB of training drive i sused irred irred contection contection contection contection contection	0 0 0 tor spec: ck and se the prev using se to "unpac ;(flag ;No, u ung algori d operati d operati d operati d operati d allocal ;set d allocal ; that ; that ; that ; and ; that ; and	<pre>shonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls cetor. The sector will be read vious call to set DMA address. cetors larger than 128 bytes, cetors larger than 128 bytes, cetor</pre>
$\begin{array}{c} 1149\\ 1150\\ 1151\\ 1152\\ 1153\\ 1154\\ 1157\\ 1156\\ 1157\\ 1158\\ 1157\\ 1158\\ 1157\\ 1158\\ 1161\\ 1162\\ 1164\\ 1162\\ 1164\\ 1165\\ 1168\\ 1167\\ 1168\\ 1167\\ 1171\\ 1172\\ 1173\\ 1174\\ 1175\\ 1177\\ 1178\\ 1179\end{array}$	FBF9 00 FBFA 00 FBFA 00 FBFB 3AF9FB FBFE B7 FBFF CA52FD FC02 32F5FB FC06 3C FC07 32F8FB FC06 3C FC07 32F8FB FC06 32F7FB	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ XRA STA STA STA STA	g\$Required: : : ect disk and to she address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; The de ; that ; up u ; thou ; unal A Unallocated\$Rec A Read\$Operation Must\$Preread\$Se	DB DB DB of training drive i sused irred irred contection contection contection contection contection	0 0 0 tor spec: ck and set the prev scheck ;(flat ;(flat ;(flat ;(flat ;(flat ;(flat ;(flat ;(flat ;(flat ;), d operati d algorid d algo	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. etors larger than 128 bytes, ck" a 128-byte sector from (if deblocking needed g was set in SELDSK call) use normal nondeblocked ithm used is such toon can be viewed L data transfer as first write to an tion block. the record count to 0 or first "write" cate that it is really a read at is to be performed i force a preread of the sector</pre>
1149 1150 1151 1152 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1177	FBF9 00 FBFA 00 FBFA 00 FBFA 00 FBFB 3AF9FB FBFE 87 FBFF CA52FD FC03 32F5FB FC03 32F5FB FC04 3C FC07 32F8FB FC0A 32F7FB FC0A 32F7FB FC0A 32F7FB	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ XRA STA STA STA STA STA	g\$Required: : : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; That ; that ; up u ; that ; up u ; that ; up u ; that ; up u ; that ; un u ;	DB DB DB of training drive i sused irred irred contection contection contection contection contection	0 0 tor spec: ck and set the prev sched set ;(flag ;(flag ;(flag ;(flag ;(flag ;(flag ;(flag ;)(flag ;(flag ;)(flag ;)(flag ;(flag ;)(<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK). ified by previous calls ector. The sector will be read vious call to set DMA address. ctors larger than 128 bytes, ck" a 128-byte sector from (if deblocking needed g was set in SELDSK call) ise normal nondeblocked ithm used is such ion can be viewed l data transfer as first write to an tion block. the record count to 0 or first "write" rate that it is really a read at is to be performed d force a preread of the sector get it into the disk buffer deblocking code into responding</pre>
$\begin{array}{c} 1149\\ 1150\\ 1151\\ 1152\\ 1153\\ 1154\\ 1155\\ 1156\\ 1157\\ 1158\\ 1157\\ 1158\\ 1160\\ 1161\\ 1162\\ 1163\\ 1164\\ 1165\\ 1167\\ 1168\\ 1167\\ 1172\\ 1173\\ 1176\\ 1177\\ 1178\\ 1177\\ 1178\\ 1177\\ 1178\\ 1180\\ 1181\end{array}$	FBF9 00 FBFA 00 FBFA 00 FBFA 00 FBFB 3AF9FB FBFE 87 FBFF CA52FD FC03 32F5FB FC03 32F5FB FC04 3C FC07 32F8FB FC0A 32F7FB FC0A 32F7FB FC0A 32F7FB	Deblockin Disk‡Type ; ; Read i ; to sel ; into t ; If rea ; debloc ; the ph READ: LDA ORA JZ XRA STA STA STA STA STA	g\$Required: : : n the 128-byte CF ect disk and to s he address specif ding from a disk king code will be ysical sector. Deblocking\$Requ A Read\$No\$Deblock ; That ; that ; up u ; that ; up u ; that ; up u ; that ; up u ; that ; un u ;	DB DB DB V/M sec set tra- ied in drive second cocki co	0 0 0 0 tor spec: ck and set the prev using set to "unpac ;Check ;flat ;No, u mg algor ;No, u ng algor ;No, u d operati d allocat d allocat d allocat d allocat d allocat s; to i, and ; to ; Fake ; as ; una	<pre>;Nonzero when a CP/M 128-byte ; sector is to be read ;Nonzero when the selected disk ; needs deblocking (set in SELDSK) ;Indicates 8" or 5 1/4" floppy ; selected (set in SELDSK).</pre>

~

1184		;		
1185		; Write	a 128-byte sector from t	he current DMA address to
1186		; the pr	eviously selected disk,	track, and sector.
1187		;		
1188				have set register C to indicate
1189				s to an already allocated allocation
1190		; block	(which means a preread o	f the sector may be needed),
1191				e the data will be written to the
1192				irst 128-byte sector of a previously
1193			cated allocation block (in which case no preread is required).
1194		;		
1195				ake place immediately. In all other
1196				from the DMA address into the disk
1197				hen circumstances force the
1198				al disk operations can therefore
1199 1200		; be red	uced considerably.	
1200		WRITE:		
	FC15 3AF9F		Deblocking\$Required	;Check if deblocking is required
1202		ORA	A	;(flag set in SELDSK call)
1204	FC19 CA4DFI		Write\$No\$Deblock	, (Trag set In SEEBSK Carry
1205	I OI / CHION		WITCHNOPPEDIOCK	
1206	FC1C AF	XRA	A	;Indicate that a write operation
1207	FC1D 32F8F		Read\$Operation	; is required (i.e. NOT a read)
1208	FC20 79	MOV	A,C	; Save the BDOS write type
1209	FC21 32E3F		Write\$Type	,
1210		CPI	Write\$Unallocated	;Check if the first write to an
1211				; unallocated allocation block
1212	FC26 C237F	D JNZ	Check\$Unallocated\$Bloc	k ;No, check if in the middle of
1213				; writing to an unallocated block
1214				;Yes, first write to unallocated
1215				; allocation block initialize
1216				; variables associated with
1217				; unallocated writes.
	FC29 3E10	MVI	A,Allocation\$Block\$Siz	
1219				; sectors and
	FC2B 32F5F1	B STA	Unallocated\$Record\$Cou	nt ; set up a count.
1221				;
1222	FC2E 21EAF		H, Selected\$Dk\$Trk\$Sec	
1223	FC31 11F1F1		D, Unallocated \$Dk\$Trk\$S	ec ; into unallocated variables
1224	FC34 CD35FI		Move\$Dk\$Trk\$Sec	
1225		; Charle		units to an unallocated
1226				write to an unallocated the unallocated record count
1227 1228			st been set to the number	r of 128-byte sectors in the
1228			tion block.	r of 120-byte sectors in the
1230		, alloca	CION BIOCK.	
1231		, Check\$lina	llocated\$Block:	
1232	FC37 3AF5FI	B LDA	Unallocated\$Record\$Cou	nt
1233	FC3A B7	ORA	Α	
1234			Request\$Preread	;No, this is a write to an
1235				; allocated block
1236				;Yes, this is a write to an
1237				; unallocated block
	FC3E 3D	DCR	A	;Count down on number of 128-byte sectors
1239				; left unwritten to in allocation block
	FC3F 32F5F	B STA	Unallocated\$Record\$Cou	int ; and store back new value.
1241				.
1242	FC42 21EAF		H, Selected\$Dk\$Trk\$Sec	;Check if the selected disk, track,
1243	FC45 11F1F			ec; and sector are the same as for
1244	FC48 CD29F		Compare\$Dk\$Trk\$Sec	; those in the unallocated block.
	FC4B C266F	C JNZ	Request\$Preread	;No, a preread is required
1246				;Yes, no preread is needed.
1247				Now is a convenient time to
1248				; update the current sector and see
1249 1250		;		; if the track also needs updating.
1250		,		;By design, Compare\$Dk\$Trk\$Sec
1251				; returns with
1253				; DE -> Unallocated\$Sector
1254	FC4E EB	XCHG		; HL -> Unallocated\$Sector
1255	FC4F 34	INR	м	;Update Unallocated\$Sector
1256	FC50 7E	MOV	A, M	;Check if sector now > maximum
1257	FC51 FE48	CPI	CPM\$Sec\$Per\$Track	; on a track
1258	FC53 DA5FF	C JC	No\$Track\$Change	;No (A < M)
1259				;Yes,

1260	FC56	3600	MVI	M, 0	Reset sector to 0
		2AF2FB	LHLD	Unallocated\$Track	;Increase track by 1
	FC5B		INX	н	
1263	FC5C	22F2FB	SHLD	Unallocated\$Track	
1264					
1265			No\$Track\$(Change:	:Indicate to later code that
1266 1267					; no preread is needed.
1267	FC5F	۸E	XRA	Α	; no prefedo is needed.
1269		32F7FB	STA	n Must\$Preread\$Sector	;Must\$Preread\$Sector=0
1270		C36EFC	JMP	Perform\$Read\$Write	, has the read poet to -o
1271		0002.0	:		
1272			, Request\$Pr	reread:	
1273	FC66	AF	XRA	A	;Indicate that this is not a write
1274	FC67	32F5FB	STA	Unallocated\$Record\$Count	; into an unallocated block.
1275	FC6A	30	INR	A	
1276	FC6B	32F7FB	STA	Must\$Preread\$Sector	;Indicate that a preread of the
1277					; physical sector is required.
1278			;		
1279			;		
1280			Perform\$Re	ead\$Write:	;Common code to execute both reads and ; writes of 128-byte sectors.
1281 1282	FC6E	A.F.	XRA	A	; writes of 120-byte sectors. ;Assume that no disk errors will
1282		32F6FB	STA	H Disk\$Error\$F1ag	; occur
1284	FCOF	321018	314	DISK#CITO(#TIAg	, occu
1285	FC72	3AEDFB	LDA	Selected\$Sector	;Convert selected 128-byte sector
1286	FC75		RAR		; into physical sector by dividing by 4
1287	FC76		RAR		,
1288		E63F	ANI	3FH	Remove any unwanted bits
1289	FC79	32EEFB	STA	Selected\$Physical\$Sector	
1290					;
1291		21E8FB	LXI	H,Data\$In\$Disk\$Buffer	;Check if disk buffer already has
1292	FC7F		MOV	A, M	; data in it.
1293	FC80	3601	MVI	M, 1	;(Unconditionally indicate that
1294					; the buffer now has data in it)
1295	FC82		ORA	A	;Did it indeed have data in it?
1296	FC83	CAA3FC	JZ	Read\$Sector\$into\$Buffer	;No, proceed to read a physical
1297 1298					; sector into the buffer.
1298				The bu	; ffer dóes have a physical sector
1300				; in i	
1301				: Note	The disk, track, and PHYSICAL
1302				: sect	or in the buffer need to be
1303				; chec	ked, hence the use of the
1304					are\$Dk\$Trk subroutine.
1305				;	
1306		11E4FB	LXI		;Check if sector in buffer is the
1307		21EAFB	LXI	H,Selected\$Dk\$Trk\$Sec	; same as that selected earlier
1308		CD24FD	CALL	Compare\$Dk\$Trk	Compare ONLY disk and track
1309	FC8F	C29CFC	JNZ	Sector\$Not\$In\$Buffer	;No, it must be read in
1310		0.0750	1.54	T- thursday the states	;Get physical sector in buffer
1311		3AE7FB 21EEFB	LDA LXI	In\$Buffer\$Sector H,Selected\$Physical\$Sec	
1312	FC98		CMP	M	;Check if correct physical sector
1313		CABIFC	JZ	Sector\$In\$Buffer	;Yes, it is already in memory
1315		CODIT O	:		,,
1316			Sector\$No	t\$In\$Buffer:	
1317					;No, it will have to be read in
1318					; over current contents of buffer
1319		3AE9FB	LDA	Must\$Write\$Buffer	;Check if buffer has data in that
1320			ORA	Α	; must be written out first
1321	FCAO	C495FD	CNZ	Write\$Physical	;Yes, write it out
1322					
1323	-	001155		or\$into\$Buffer:	c ;Set in buffer variables from
1324	FCA3	CD11FD	CALL	Set\$In\$Buffer\$Dk\$Trk\$Se	; selected disk, track, and sector
1325					; to reflect which sector is in the
1326 1327					: buffer now
1328	FCAA	3AF7FB	LDA	Must\$Preread\$Sector	; In practice, the sector need only
1329	FCA9		ORA	A	; be physically read in if a preread
1330					; is required
1331	FCAA	C49AFD	CNZ	Read\$Physical	;Yes, preread the sector
1332	FCAD		XRA	A	Reset the flag to reflect buffer
1333		32E9FB	STA	Must\$Write\$Buffer	; contents.
1334			1		· · · · · · · · · · · · · · · · · · ·
1335			Sector\$In	\$Buffer: ;Select	ed sector on correct track and

Figure 6-4. (Continued)

٦

,

1336					. disk is already in the buffer
1337					; disk is already in the buffer. ;Convert the selected CP/M (128-byte)
1338					
1339					; sector into a relative address down
					; the buffer.
1340		3AEDFB	LDA		;Get selected sector number
1341	FCB4	E603	ANI	Sector\$Mask	;Mask off only the least significant bits
1342	FCB6 FCB7	6F	MOV	L,A	;Multiply by 128 by shifting 16-bit value
1343	FCB7	2600	MVI	н,о	; left 7 bits
1344	FCB9	29	DAD	н	;* 2
1345	FCBA		DAD	н	;* 4 ~
1346	FCBB		DAD	Ĥ	;* 8
1347	FCBC	20	DAD	Н	;* 16
1348	FCBD		DAD	H	;* 32
1349	FCBE		DAD	н	;* 64
1350	FCBF	29	DAD	н	;* 128
1351			;		-
1352	FCC0	1133F6	LXI	D,Disk\$Buffer	;Get base address of disk buffer
1353	FCC3	19	DAD	D	;Add on sector number * 128
1354					;HL -> 128-byte sector number start
1355					; address in disk buffer
1356	FCC4	FB	XCHG		;DE -> sector in disk buffer
1357		2A63FB	LHLD	DMA\$Address	;Get DMA address set in SETDMA call
1358	FCC8	FB	XCHG		Assume a read operation, so
1359			ACIIC		; DE -> DMA address
1360					
	F000	0510		6 100/0	; HL -> sector in disk buffer
1361	FCC9	UE10	MVI	C,128/8	;Because of the faster method used
1362					; to move data in and out of the
1363					; disk buffer, (eight bytes moved per
1364					; loop iteration) the count need only
1365					; be 1/8th of normal.
1366					At this point -
1367					; C = loop count
1368					; DE -> DMA address
1369					; HL -> sector in disk buffer
1370	ECCR	3AF8FB	LDA	Read\$Operation	;Determine whether data is to be moved
1370	FCCE		ORA		
1371		C2D7FC	JNZ		; out of the buffer (read) or into the
	FUUF	C2D/FC	ONZ	Buffer\$Move	; buffer (write)
1373					;Writing into buffer
1374		-			;(A must be () get here)
1375	FCD2		INR	A	;Set flag to force a write
1376		32E9FB	STA	Must\$Write\$Buffe	er ; of the disk buffer later on.
1377	FCD6	EB	XCHG		;Make DE -> sector in disk buffer
1378					; HL -> DMA address
1379			;		
1380					
1381			, Buffer\$Move	•	;The folowing move loop moves eight bytes
1382			Darrer #110V	- •	; at a time from (HL) to (DE), C contains
1383					
		76	MOUL		; the loop count.
1384	FCD7	10	MOV	A,M	;Get byte from source
1385	FCD8		STAX	D	Put into destination
1386	FCD9	13	INX	D	;Update pointers
1387	FCDA		INX	н	
1388	FCDB	7E	MOV	A, M	;Get byte from source
1389	FCDC FCDD	12	STAX	D	Put into destination
1390	FCDD	13	INX	D	;Update pointers
1391	FCDE	23	INX	ň	
1392	FCDF	7F	MOV	A, M	;Get byte from source
1393	FCEO	12	STAX	D	
	FOET	10			;Put into destination
1394	FCE1		INX	D	;Update pointers
1395	FCE2		INX	н	
1396	FCE3		MOV	A, M	;Get byte from source
1397	FCE4		STAX	D	;Put into destination
1398	FCE5	13	INX	D	;Update pointers
1399	FCE6	23	INX	н	
1400	FCE7	7E	MOV	A, M	;Get byte from source
1401	FCE8		STAX	D	;Put into destination
1402	FCE9	13	INX	Ď	;Update pointers
1403	FCEA		INX	Ĥ	, ., ,
1403	FCEB		MOV	A. M	Gat byta from course
					;Get byte from source
1405	FCEC		STAX	D	;Put into destination
1406	FCED		INX	D	;Update pointers
1407	FCEE		INX	н	
1408	FCEF		MOV	Α,Μ	;Get byte from source
1409	FCFO	12	STAX	D	;Put into destination
1410	FCF1		INX	D	;Update pointers
	· -· -				· · · · · · · · · · · · · · · · · · ·

1411	FCF2	23	INX	н	
1412	FCF3	7F	MOV	A, M	;Get byte from source
1413	FCF4		STAX	D	;Put into destination
	FUF4	12			
1414	FCF5	13	INX	D	;Update pointers
1415	FCF6	23	INX	н	
1416					
1417	FCF7	OD	DCR	С	;Count down on loop counter
1418	ECE9	C2D7FC	JNZ	Buffer\$Move	Repeat until CP/M sector moved
	1010	CED/I C	ONE	barrer thore	inepeat antir of the sector mored
1419					,
1420	FCFB	3AE3FB	LDA	Write\$Type	;If write to directory, write out
1421	FCFE	EE01	CPI	Write\$Directory	; buffer immediately
			LDA		
1422		3AF6FB		DISKAErrorariag	;Get error flag in case delayed write or read
1423	FD03	CO	RNZ		Return if delayed write or read
1424					•
1425	FD04	87	ORA	A	Check if any disk errors have occurred
				M	
1426	FD05	CO	RNZ		;Yes, abandon attempt to write to directory
1427					•
1428	ED04	AF	XRA	•	Clear flag that indicates buffer must be
	FD06	MF			
1429		32E9FB	STA	Must\$Write\$Buff	
1430	FDOA	CD95FD	CALL	Write\$Physical	;Write buffer out to physical sector
1431		3AF6FB	LDA		Return error flag to caller
				DISK#EFFOF#FIA9	freturn error i lag to caller
1432	FD10	C9	RET		
1433			;		
1434					
					المسترية المسترية المستقل والمسترية المسترية المسترية
1435			⇒et\$in\$Buf	fer\$Dk\$Trk\$Sec:	;Indicate selected disk, track, and
1436					; sector now residing in buffer
1437	5011	3AEAFB	LDA	Selected\$Disk	
1438	FD14	32E4FB	STA	In\$Buffer\$Disk	
1439					
1440	ED17	2AEBFB	LHLD	Selected\$Track	
1441	FUIA	22E5FB	SHLD	In\$Buffer\$Track	
1442					
1443	FDID	3AEEFB	LDA	Selected\$Physic	al\$Sector
				In\$Buffer\$Sector	
1444	FD20	32E7FB	STA	Insputterssector	
1445					
1446	FD23	60	RET		
	1020	U /			
1447			;		
1448			Compare\$Dk1	\$Trk:	;Compares just the disk and track
1449					; pointed to by DE and HL
	FD24	0503	MVI	C.3	:Disk (1), track (2)
1450					
1451	FD26	C32BFD	JMP	Compare\$Dk\$Trk\$	Sec\$Loop ;Use common code
1452					
			Compare\$Dk	tTuldCoo.	Compares the disk track and sector
1453			Compareaux	PITKPORC:	Compares the disk, track, and sector
1454					; variables pointed to by DE and HL
1455	FD29	0E04	MVI	C,4	;Disk (1), track (2), and sector (1)
1456				<pre>\$Trk\$Sec\$Loop:</pre>	
1457	FD2B	1A	LDAX	D	;Get comparitor
1458	FD2C	BF	CMP	M	;Compare with comparand
	FD2D		RNZ		;Abandon comparison if inequality found
1459				-	
1460	FD2E		INX	D	;Update comparitor pointer
1461	FD2F	23	INX	н	;Update comparand pointer
1462	FD30		DCR	c	;Count down on loop count
1463	FD31		RZ	_	;Return (with zero flag set)
1464	FD32	C32BFD	JMP	Compare\$Dk\$Trk\$	Sec\$Loop
1465			;		
1466					
			1 mm -		Manual Aba dialy Assals and against
1467			Move\$Dk\$Tr	K#Sec:	;Moves the disk, track,and sector
1468					; variables pointed at by HL to
1469					; those pointed at by DE
1470	FD35	0E04	MVI	C,4	;Disk (1), track (2), and sector (1)
1471			Move\$Dk\$Tr	k\$Sec\$Loop:	
1472	FD37	75	MOV	A, M	;Get source byte
1473	FD38		STAX	D	;Store in destination
1474	FD39	13	INX	D	;Update pointers
1475	FD3A		INX	Ĥ	
			DCR	C	:Count down on byte count
1476	FD3B			U	
1477	FD3C		RZ		;Return if all bytes moved
1478		C337FD	JMP	Move\$Dk\$Trk\$Sec	\$Loop
1479			;		
1480			;		
1482			;		
1483			. There -	re two "emart" d	isk controllers on this system, one
1484					tte drives, and one for the 5 1/4"
1485			; mini-di	skette drives.	
1486					
			, The	trollove sus HL-	rd-wired" to monitor certain locations
1487			; ine con	troilers are "ha	rd-wired" to monitor certain locations

Figure 6-4. (Continued)

1488			• in memo	ry to detect whe	n thay a	re to pe	rform como dick			
1489			: operati	on. The 8" contr	oller mo	are to perform some disk monitors location 0040H, and				
1490						location 0045H. These are				
1491						es. If the most significant				
1492				a disk control b						
1493						respective control bytes.				
1494			; This wo	rd must contain	the addr	ess of a	valid disk control			
1495							ration to be performed.			
1496			;							
1497			; Once th	e operation has	been com	pleted.	the controller resets			
1498			; its dis	k control byte t	0 00H. T	his indi	cates completion			
1499				disk driver code						
1500			;							
1501			; The con	troller also set	s a retu	rn code	in a disk status block			
1502				ntrollers use th						
1503			; If the	first byte of th	is statu	s block	is less than 80H, then			
1504							ple BIOS, no further details			
1505			; of the	status settings	are rele	vant. No	te that the disk controller			
1506			; has bui	lt-in retry logi	c rea	ds and w	rites are attempted ten			
1507			; times b	efore the contro	ller ret	urns an i	error.			
1508			;							
1509			; The dis	k control table	layout i	s shown l	below. Note that the			
1510							trol tables to be			
1511							disk operations can			
1512							is not used. However,			
1513				troller requires						
1514							the main control bytes			
1515 1516			; in orde	r to indicate th	e end of	the cha	1n.			
1516	0040	_	7 Déslation	-149	EQU	40H				
1518	0040		Disk\$Contr				;8" control byte			
1518	0041 -	-	Command\$B1	006898	EQU	41H	;Control table pointer			
1520	0043 :	-	; Disk\$Statu	c#Plook	EQU	43H	;8" AND 5 1/4" status block			
1521	0043 -	-		S#BIOCK	EQU	400	38 MND 3 174 Status Diock			
1522	0045	-	, Disk\$Contr	01\$5	EQU	45H	;5 1/4" control byte			
1523	0046		Command\$B1		FOU	46H	Control table pointer			
1524	0040	_	;	0000	Eao	400	control table pointer			
1525			,							
1526			; Floppy	Disk Control Tab	105					
1527			1 1100007	Disk control lab	163					
1528	FD40 (00	, Floppy\$Com	mand:	DB	0	:Command			
1529	0001 -		Floppy\$Rea		EQU	õ1H	y commune			
1530	0002 -		Floppy\$Wri		EQU	02H				
1531	FD41 (Floppy\$Uni		DB	0	;Unit (drive) number = 0 or 1			
1532	FD42 (Floppy\$Hea		DB	ò	;Head number = 0 or 1			
1533	FD43 (Floppy\$Tra		DB	ō	;Track number			
1534	FD44 (Floppy\$Sec		DB	ò	;Sector number			
1535	FD45 (Floppy\$Byt		DW	ō	Number of bytes to read/write			
1536	FD47 (Floppy\$DMA		DW	0	Transfer address			
1537	FD49 (0000		t\$Status\$Block:	D₩	0	Pointer to next status block			
1538							; if commands are chained.			
1539	FD4B (0000	Floppy\$Nex	t\$Control\$Locatio	on: DW	0	Pointer to next control byte			
1540							; if commands are chained.			
1541			;							
1542			;							
1543			;							
1544			Write\$No\$D	eblock:			contents of disk buffer to			
1545							ect sector.			
1546	FD4D 3	3E02	MVI	A,Floppy\$Write\$			ite function code			
1547	FD4F (C354FD	JMP	Common\$No\$Deblo	ck		common code			
1548			Read\$No\$De	block:			reviously selected sector			
1549			Mai		•		disk buffer.			
1550	FD52 3	BEO1	MVI	A, Floppy\$Read\$C	ode	;Get rea	ad function code			
1551			Common\$No\$							
1552	FD54 3	3240FD	STA	Floppy\$Command			nction code			
1553							ocked command table			
1554	FD57 2			H, 128		per secto	or			
1555	FD5A 2		SHLD	Floppy\$Byte\$Cour			has head 0			
1556	FD5D A		XRA	A Elenny#Head	10. 110	-bh outh	has head O			
1557	FD5E 3	0242FU	STA	Floppy\$Head						
1558 1559	FD61 3		LDA	Selected\$Disk	, 		roller only has information			
1560	PD01 3	DEALD	LUM	Serectenablek			nd 1 so Selected\$Disk must			
1560						onverted	NG I SO GELECIEU#DISK MUSI			
1562	FD64 8	5601	ANI	01H		nto 0 or	1			
1563	FD66 3	3241FD	STA	Floppy\$Unit		it number				
1000										

Figure 6-4. (Continued)

1564					,		
1565		3AEBFB	LDA	Selected\$Track			
1566	FD6C	3243FD	STA	Floppy\$Track	;Set track	< number	·
1567				0-1	;		
1568 1569		3AEDFB 3244FD	LDA STA	Selected#Sector Floppy#Sector	;Set secto	w numbe	
1570	FUIZ	324450	SIM	FIOPPy#Sector	soet secto	or numbe	2 T
1571	FD75	2A63FB	LHLD	DMA\$Address	Transfer	direct)	ly between DMA address
1572		2247FD	SHLD	Floppy\$DMA\$Addre			controller.
1573					;		
1574					;The disk	contro!	ller can accept chained
1575 1576					; 015K CC	ontrol (tables, but in this case, used, so the "Next" pointers
1577					: must be	pointe	ed back at the initial
1578							in the base page.
1579		214300	LXI	H,Disk\$Status\$B		1	Point next status back at
1580	FD7E	2249FD	SHLD	Floppy\$Next\$Stat	tus\$Block	1	; main status block
1581	FROI	214000	LXI	H.Disk\$Control\$8			Point next control byte
1582 1583		214000 224BFD	SHLD	Floppy\$Next\$Con			back at main control byte
1584	1 004	224010	OTICD.	1100074462000	() OIPEOCAC.		back at main control byte
1585		2140FD	LXI	H,Floppy\$Command			Point controller at control table
1586	FD8A	224100	SHLD	Command\$Block\$8			
1587				11 04-140	•	1	
1588 1589		214000 3680		H,Disk\$Control\$8 M.80H	0		Activate controller to perform operation.
1590		C3F7FD	JMP	Wait\$For\$Disk\$C	omplete		, operation.
1591			-,				
1592			;				
1593			;				
1594 1595			Write\$Phys	ical:	• 1	drite co	ontents of disk buffer to
1596			Wille+1175	ICal.			ct sector.
1597	FD95	3E02	MVI	A.Floppy\$Write\$0			te function code
1598		C39CFD	JMP	Common \$Physical	;(Go to co	ommon code
1599			Read\$Physi	cal:	; F	Read pre	eviously selected sector
1600			MI 17				disk buffer.
1601 1602	FD9A	3E01	MVI	A,Floppy\$Read\$C	ode ;	set read	d function code
1602			, Common\$Phy	sical:			
1604	FD9C	3240FD		Floppy\$Command	; (Set com	mand table
1605							
1606			;				
1607		SAFAFB	LDA CPI	Disk\$Type Floppy\$5			k type (set in SELDSK) it is a 5 1/4" Floppy
1608 1609	FDA2	CAADFD	JZ	Correct\$Disk\$Typ		res	10 15 a 5 1/4 (10pp)
1610		3E01	MVI	A, 1			icate disk error
1611	FDA9	32F6FB	STA	Disk\$Error\$Flag			
1612	FDAC	C9	RET				4/-1
1613 1614			Correct\$Di	skalype:		set up (disk control table
1614	FDAD	3AE4FB	LDA	In\$Buffer\$Disk	; : (Convert	disk number to 0 or 1
1616	FDBO		ANI	1			isk controller
1617		3241FD	STA	Floppy\$Unit			1
1618							
1619 1620		2AE5FB	LHLD MOV	In\$Buffer\$Track A.L			track number his is single byte value
1620	FDB8	70 3243FD	STA	A,L Floppy\$Track			e controller.
1621	1009	JETUTU	SIM	· LOPP/FILECK	,	tor en	
1623							tor must be converted into a
1624							number and sector number.
1625					1		rs 0 - 8 are head 0, 9 - 17
1626 1627	EDBC	0600	MVI	B,0	7	are he Assume l	
1628		3AE7FB	LDA	In\$Buffer\$Sector			sical sector number
1629	FDC1		MOV	C,A	; 9	Save co	py in case it is head O
1630	FDC2	FE09	CPI	9		Check i	
1631		DACBFD	JC	Head\$0		Yes it :	
1632 1633	FDC7	D609	SUI	9			ify sector number back e 0 - 8 range.
1633	FDC9	4F	MOV	C,A			tor in B
1634	FDCA		INR	B		Set to I	
1636		- •	Head\$0:				
1637	FDCB	78	~ MOV	A,B	; (Set head	d number
1638	FDCC	3242FD	STA	Floppy\$Head			
1639	FDCF	/9	MOV	A,C	15	bet sec	tor number

	1640	FDDO	30	INR	Α	; (physical sectors start at 1)
	1641		3244FD	STA	Floppy\$Sector	
	1642					;
	1643		210002	LXI	H,Physical\$Sector\$Size	;Set byte count
	1644	FDD7	2245FD	SHLD	Floppy\$Byte\$Count	
	1645					;
	1646		2133F6	LXI	H,Disk\$Buffer	;Set transfer address to be
	1647	FDDD	2247FD	SHLD	Floppy\$DMA\$Address	; disk buffer 🗠
	1648 1649					;
	1650					As only one control table is in
	1651					; use, close the status and busy ; chain pointers back to the
	1652					; main control bytes.
	1653	FDEO	214300	LXI	H,Disk\$Status\$Block	, warn control bytes
	1654		2249FD	SHLD	Floppy\$Next\$Status\$Bloc	k
	1655	FDE6	214500	LXI	H,Disk\$Contro1\$5	
	1656	FDE9	224BFD	SHLD	Floppy\$Next\$Control\$Loc-	ation
	1657					
	1658	FDEC	2140FD	LXI	H,Floppy\$Command	;Set up command block pointer
	1659	FUEF	224600	SHLD	Command\$Block\$5	
	1660		-			
	1661 1662	FDF2	214500 3680	LXI MVI	H,Disk\$Contro1\$5 M,80H	Activate 5 1/4" disk controller
	1663	1.01.0	0000	1147	1,001	
	1664			Wait\$For\$D	isk\$Complete:	;Wait until Disk Status Block indicates
	1665					; operation complete, then check
	1666					; if any errors occurred.
	1667					;On entry HL -> disk control byte
	1668	FDF7		MOV	A, M	;Get control byte
	1669	FDF8		ORA		• ··· ····
	1670 1671	FUFY	C2F7FD	JNZ	Wait\$For\$Disk\$Complete	Operation still not yet done
	1672	EDEC	344300	LDA	Disk\$Status\$Block	; ;Complete now check status
	1673	FDFF		CPI	80H	Check if any errors occurred
	1674		DA09FE	JC	Disk\$Error	Yes
	1675	FE04		XRA	A	;No
	1676		32F6FB	STA	Disk\$Error\$Flag	Clear error flag
	1677	FE08	C9	RET		
	1678			Disk\$Error		
	1679	FE09		MVI	A, 1	;Set disk-error flag nonzero
	1680 1681	FEOE	32F6FB	STA RET	Disk\$Error\$Flag	
	1682	FEVE	.,	;		
	1683			;		
	1684		671	;		
	1685			; Disk com	ntrol table images for w	arm boot
	1686			;		
	1687			Boot\$Contr		
	1688	FEOF		DB	1	Read function
	1689	FE10		DB	0	;Unit (drive) number
	1690 1691	FE11 FE12		DB DB	0	;Head number
	1692	FE12	02	DB	2	;Track number ;Starting sector number
l	1693	FE14		DW	8×512	;Number of bytes to read
	1694	FE16		DW	CCP\$Entry	Read into this address
	1695	FE18	4300	DW	Disk\$Status\$Block	Fointer to next status block
	1696	FE1A	4500	DW	Disk\$Contro1\$5	Pointer to next control table
	1697			Boot\$Contro		
	1698	FE1C		DB	1	Read function
	1699 1700	FEID		DB DB	0	;Unit (drive) number
	1700	FE1E FE1F		DB	1	;Head number ;Track number
	1702	FE20		DB	1	;)rack number ;Starting sector number
	1703	FE21		DW	3*512	Number of bytes to read
	1704	FE23	00F0	DW	CCP\$Entry + (8*512)	Read into this address
L	1705	FE25	4300	DW	Disk\$Status\$Block	Pointer to next status block
1	1706	FE27		DW	Disk\$Contro1\$5	Pointer to next control table
1	1707					
1	1708			;		
l	1709			;		
	1710 1711			; WBOOT:	;Warm boot entr	×.
	1712			AB0011		
	1713					the CCP and BDOS must be reloaded In this BIOS, only the 5 1/4"
L	1714					ll be used. Therefore this code
L						

1715 is hardware specific to the controller. Two ; prefabricated control tables are used. 1716 SP.80H 1717 FE29 318000 LXI FE2C 110FFE FE2F CD3BFE D,Boot\$Control\$Part1 ;Execute first read of warm boot 1718 1719 LXI CALL Warm\$Boot\$Read ;Load drive 0, track 0, head 0, sectors 2 to 8 1720 ; ;Execute second read 1721 FE32 111CFE D,Boot\$Contro1\$Part2 LXI FE35 CD3BFE 1722 CALL Warm\$Boot\$Read ;Load drive 0, track 0, 1723 ; head 1, sectors 1 - 3 1724 FE38 C340F8 JMP Enter\$CPM ;Set up base page and enter CCP 1725 1726 Warm\$Boot\$Read: ;On entry, DE -> control table image ;This control table is moved into 1727 ; the main disk control table and 1728 ; then the controller activated. ;HL -> actual control table ;Tell the controller its address ;Move the control table image 1729 FE3B 2140FD FE3E 224600 LXI H,Floppy\$Command 1730 1731 SHLD Command\$Block\$5 into the control table itself 1733 1734 FE41 OEOD MVI ;Set byte count C.13 1735 Warm\$Boot\$Move: 1736 FE43 1A LDAX D ;Get image byte 1737 FE44 77 MOV M.A ;Store into actual control table 1738 FE45 23 TNX н ;Update pointers FE46 13 FE47 OD 1739 INX D 1740 ;Count down on byte count ;Continue until all bytes moved DCR. C 1741 FE48 C243FE JNZ Warm\$Boot\$Move 1742 1743 FE4B 214500 LXI H.Disk\$Contro1\$5 Activate controller 1744 FE4E 3680 MVI M, 80H 1745 Wait\$For\$Boot\$Complete: 1746 FE50 7E MOV A,M ;Get status byte 1747 FE51 B7 ORA ;Check if complete 1748 FE52 C250FE JNZ Wait\$For\$Boot\$Complete ;No 1748 1749 1750 1751 1752 1753 1754 ;Yes, check for errors FE55 3A4300 Disk\$Status\$Block FE58 FE80 CPI 80H FESA DASEFE JC Warm\$Boot\$Error ;Yes, an error occurred FE5D C9 RET 1755 Warm\$Boot\$Error: LXI H,Warm\$Boot\$Error\$Message 1756 FE5E 2167FE 1757 FE61 CD33F8 CALL Display\$Message 1758 1759 FE64 C329FE JMP WBOOT ;Restart warm boot 1760 Warm\$Boot\$Error\$Message: CR,LF, 'Warm Boot Error - retrying...', CR, LF, 0 1761 FE67 0D0A576172 DB 1762 ; 1763 ; FE89 END :Of simple BIOS listing 1764

Figure 6-4. (Continued)

The Major Steps Building Your First System Using SYSGEN to Write CP/M to Disk Using DDT to Build the CP/M Memory Image The CP/M Bootstrap Loader Using MOVCPM to Relocate the CCP and BDOS Putting It All Together



Building a New CP/M System

This chapter describes how to build a version of CP/M with your own BIOS built into it. It also shows you how to put CP/M onto a floppy disk and how to write a bootstrap loader to bring CP/M into memory.

The manufacturer of your computer system plays a significant role in building a new CP/M system. Several of CP/M's utility programs may be modified by manufacturers to adapt them to individual computer systems. Unfortunately, not all manufacturers customize these programs. You should therefore invest some time in studying the documentation provided with your system to see what and how much customizing may have already been done. You should also assemble and print out listings of all assembly language source files from your CP/M release diskette.

It is impossible to predict the details of customization and special procedures that the manufacturer may have installed on your particular system. Therefore, this chapter describes first the overall mechanism of building a CP/M system, and second the details of building a CP/M system around the example BIOS shown in the previous chapter as Figure 6-4.

The Major Steps

Building a new CP/M system consists of the following major steps:

- Create a new or modified BIOS with the appropriate device drivers in it. Assemble this so that it will execute at the top end of memory (by using an *origin* statement (ORG) to set the location counter).
- Create new versions of the CCP and BDOS with all addresses in the instructions changed so that they will be correctly located in memory just below the new BIOS. Digital Research provides a special utility called MOVCPM to do this.
- Create or modify a CP/M bootstrap loader that will be loaded by the firmware that executes when you first switch on your computer (or press the RESET button). Normally, the CP/M bootstrap loader executes in the low-address end of memory. The exact address and the details of any hardware initialization that it must perform will depend entirely on your particular computer system.
- Using Digital Research standard utility programs, bring the bootstrap loader, the CCP and BDOS, and the BIOS together in the low part of memory. Then write this new version of CP/M onto a disk in the appropriate places. Again, depending on the design of your computer system, you may be able to use the standard utility program, SYSGEN, to write the entire CP/M *image* onto disk. Otherwise you may have to write a special program to do this.

When CP/M is already running on your computer system and you want to add new features to the BIOS, all you need to do is change the BIOS and rebuild the system. The CCP and BDOS will need to be moved down in memory if the changes expand the BIOS significantly. If this happens, you will have to make minor changes in the bootstrap loader so that it reads the new CP/M image into memory at a lower address and transfers control to the correct location (the first instruction of the BIOS jump vector).

Building Your First System

The first time that you build CP/M, it is a good idea to make no changes to the BIOS at all. Simply reassemble the BIOS source code and proceed with the system build. Then, if the new system does not run, you know that it must be something in the procedure you used rather than any new features or modification to the BIOS

source code. Changes in the BIOS could easily obscure any problems you have with the build procedure itself.

The Ingredients

To build CP/M, you will need the following files and utility programs:

- The assembly language source code for your BIOS. Check your CP/M release diskette for a file with a name like CBIOS.ASM (Customized Basic Input/Output System). Some manufacturers do not supply you with the source code for their BIOS; it may be sold separately or not released at all. If you cannot get hold of the source code, the only way that you can add new features to the BIOS is by writing the entire BIOS from scratch.
- The source code for the CP/M bootstrap loader. This too may be on the release diskette or available separately from your computer's manufacturer.
- The Digital Research assembler, which converts source code into machine language in hexadecimal form. This program, called ASM.COM, will be on your CP/M release diskette. Equivalent assemblers, such as Digital Research's macro-assemblers MAC and RMAC or Microsoft's M80, can also be used.
- The Digital Research utility called MOVCPM, which prepares a memory image of the CCP and BDOS with all addresses adjusted to the right values.
- The Digital Research debugging utility, called DDT (Dynamic Debugging Tool), or the more enhanced version for the Z80 CPU chip, ZSID (Z80 Symbolic Interactive Debugger). DDT is used to read in the various program files and piece together a memory image of the CP/M system.
- The Digital Research utility program SYSGEN. This writes the composite memory image of the bootstrap, CCP, BDOS, and BIOS onto the disk. SYSGEN was designed to work on floppy disk systems. If your computer uses a hard disk, you may have a program with a name like PUTCPM or WRITECPM that performs the same function.

The Ultimate Goal

In Figure 6-4, lines 0044 to 0065, you can see the equates that define the base addresses for the CCP, the BDOS, and the BIOS. Figure 7-1 shows how the top of memory will look when this version of CP/M has been loaded into memory.

Life would be simple if you could build this image in memory at the addresses shown and write the image out to disk. Building this image, however, would probably overwrite the version of CP/M that you were operating since it too lives at the top of memory. Therefore, the goal is to create a replica of this image lower down in memory, but with all the instruction addresses set to *execute* at the addresses shown in Figure 7-1.

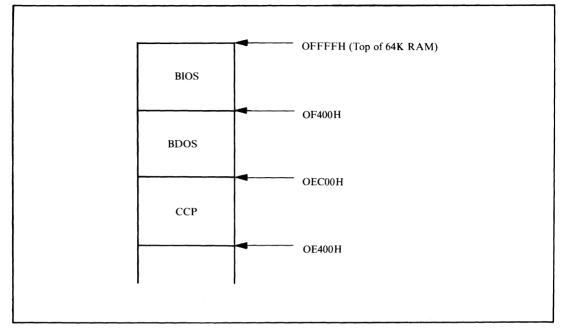


Figure 7-1. Memory layout of CP/M

Using SYSGEN to Write CP/M to Disk

The SYSGEN utility writes a memory image onto a specified logical disk. It can use a memory image that you arrange to be in memory before you invoke SYSGEN, or you can direct SYSGEN to read in a disk file that contains the image. You can also use SYSGEN to transport an existing CP/M system from one diskette to another by directing it to load the CP/M image from one diskette into memory and then to write that image out to another diskette.

Check the documentation supplied by your computer's manufacturer to make sure that you can use SYSGEN on your system. SYSGEN, as released by Digital Research, is constructed to run on 8-inch, single-sided, single-density diskettes. If your system does not use these standard diskettes, SYSGEN must be customized to your disk system.

When SYSGEN loads a CP/M image into memory, it will place the bootstrap, CCP, BDOS, and BIOS at the predetermined addresses shown in Figure 7-2, regardless of where this CP/M originated.

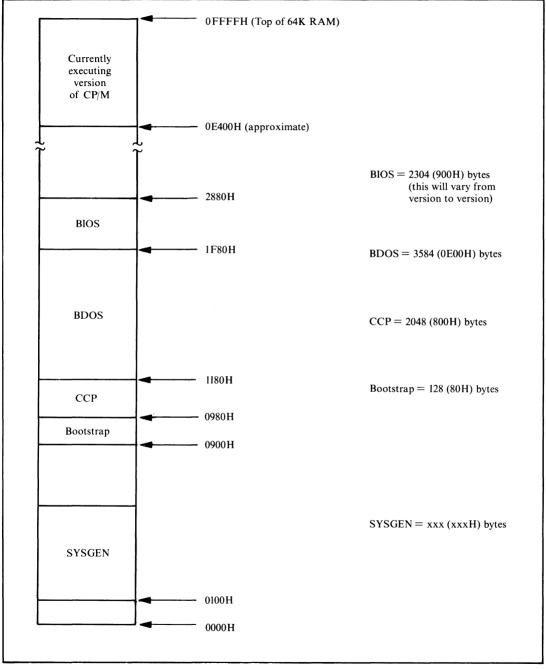


Figure 7-2. SYSGEN's memory layout

You can see that the *relative* arrangement between the components has not changed; the whole image has simply been moved down in memory well below the currently executing version of CP/M. The bootstrap has been added to the picture just beneath the CCP.

The SYSGEN utility writes this image onto a floppy diskette starting at sector 1 of track 0 and continuing to sector 26 on track 1. Refer back to Figure 2-2 to see the layout of CP/M on a standard 8-inch, single-sided, single-density diskette.

If you request SYSGEN to read the memory image from a file (which you do by calling SYSGEN with the file name on the same line as the SYSGEN call), then SYSGEN presumes that you have previously created the correct memory image and saved it (with the SAVE command). SYSGEN then skips over the first 16 sectors of the file so as to avoid overwriting itself.

Here is an example of how to use SYSGEN to move the CP/M image from one diskette to another:

```
A><u>SYSGEN<CR></u>

SYSGEN VER 2.0

SOURCE DRIVE NAME (OR RETURN TO SKIP) <u>A</u>

SOURCE ON A:, THEN TYPE RETURN <u><er></u>

FUNCTION COMPLETE

DESTINATION DRIVE NAME (OR RETURN TO REBOOT) <u>B</u>

DESTINATION ON B: THEN TYPE RETURN <u><er></u>

FUNCTION COMPLETE

DESTINATION DRIVE NAME (OR RETURN TO REBOOT) <u><er></u>

A>_
```

As you can see, SYSGEN gives you the choice of specifying the source drive name or typing CARRIAGE RETURN. If you enter a CARRIAGE RETURN, SYSGEN assumes that the CP/M image is already in memory. Note that you need to call up SYSGEN only once to write out the same CP/M image to more than one disk.

A larger than standard BIOS can cause difficulties in using SYSGEN. The standard SYSGEN format only allows for six 128-byte sectors to contain the BIOS, so if your BIOS is larger than 768 (300H) bytes, it will be a problem. The CP/M image will not fit on the first two tracks of a standard 8-inch diskette.

Nowadays it is rare to find an 8-inch floppy diskette system where you must load CP/M from a single-sided, single-density diskette. Most systems now use double-sided or double-density diskettes as the normal format, but can switch to single-sided, single-density diskettes to interchange information with other computer systems.

Because there is no "standard" format for 8-inch, double-sided and doubledensity diskettes, you probably won't be able to read diskettes written on systems of a different make or model. Therefore, you need only be concerned about using a disk layout that will keep your disks compatible with other machines that are exactly the same as yours.

This is also true if you have 5 1/4-inch diskettes. There is no industry standard for these either, so your main consideration is to place the file directory in the same

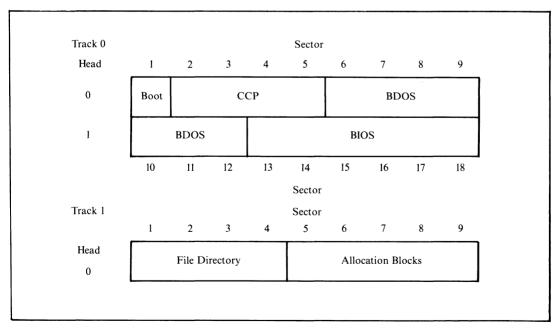
place as it will be on diskettes written by other users of your model of computer. You must also be sure to use the same sector skewing. Otherwise, you will get a garbled version whenever you try to read files originating on other systems.

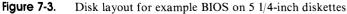
With the higher capacity diskettes, you can reserve more space to hold the CP/M image on the diskette. For example, in the case of the BIOS shown in Figure 6-4, the CP/M image is written to a 5 1/4-inch, double-sided, double-density diskette using 512-byte sectors. Figure 7-3 shows the layout of this diskette. Note that the bootstrap loader is placed in a 512-byte sector all by itself. Doing so makes the bootstrap code and warm boot code in the BIOS much simpler.

The memory image must be altered to reflect the fact that the bootstrap now occupies an entire 512-byte sector. Rather than change all of the addresses, the bootstrap is loaded into memory 384 (180H) bytes lower, so that it ends at the same address as before. Figure 7-4 shows the revised memory image.

Writing a PUTCPM Utility

Because the example system uses 5 1/4-inch floppy diskettes with 512-byte sectors, the standard version of SYSGEN cannot be used to write the CP/M image onto a diskette. You will have to use a functional replacement provided by your computer's manufacturer or develop a small utility program to do the job.





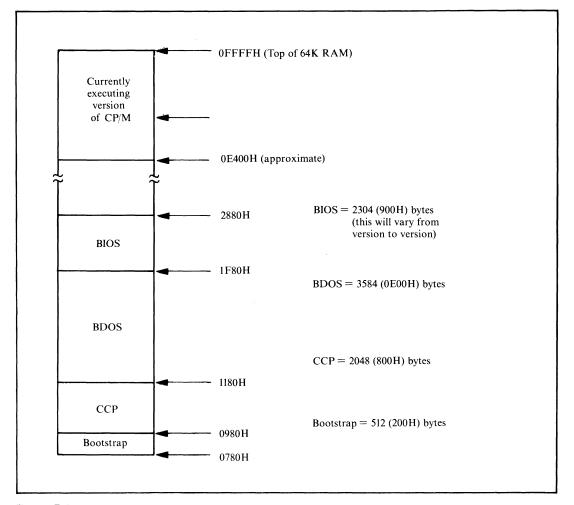


Figure 7-4. Addresses for example BIOS image

Figure 7-5 shows an example of such a program. It is written in a generalpurpose way, so that you may be able to use it for your system by changing the equates at the front of the program to reflect the specifics of your disk drives.

Note that there are two problems to be solved. First, the area of the disk on which the CP/M image resides cannot be accessed by the BDOS, as it is outside the file system area on the disk. Second, it is rare to write the CP/M image onto the disk with any kind of sector skewing; to do so would slow down the loading process. In any case, skewing would be redundant, since the loader is doing no processing other than reading the disk and can therefore read the disk without skewing.

```
This program writes out the CP/M cold boot loader,
CCP, BDOS, and BIOS to a floppy diskette. It runs
under CP/M as a normal transient program.
                :
                ;
                :
3130 =
                Version
                                 FOIL
                                          1011
                                                   ;Equates used in the sign-on
                                                   ; message
                                 FOU
                                          1071
3730 =
                Month
                                 EQU
                                          1241
3432 =
                Day
3238 =
                                 EQU
                                           ·82 ·
                Year
                         The actual PUTCPMF5.COM program consists of this code,
                $
                         plus the BOOTF5.HEX, CCP, BDOS, and BIOS.
                :
                ;
                         When this program executes, the memory image should
                         look like this:
                               Component
                                               Base Address
                                 BIOS
                                                   1F80H
                                 BDOS
                                                   1180H
                                 CCP
                                                   0980H
                .
                                 BOOTF5
                                                   0780H
                :
                         The components are produced as follows:
                3
                ÷
                                 BIOS.HEX
                                                  By assembling source code
                                                  From a CPMnn.COM file output
by MOVCPM and SAVEd on disk
                                 BDOS )
                                 CCP )
                                 BOOTF5.HEX
                                                  By assembling source code
                2
                        The components are pieced together using DDT with the
                ;
                        following commands:
                :
                                 DDT CPMnn.COM
                ;
                                 IPUTCPMF5.HEX
                .
                                 R
                                                           (Reads in this program)
                2
                                 IBOOTF5.HEX
                                 R680
                                                           (Reads in BOOT at 0780H)
                                 IBIOS.HEX
                                 R2980
                                                           (Reads in BIOS at 1F80H)
                                 GO
                                                           (Exit from DDT)
                                 SAVE 40 PUTCPMF5.COM
                                                           (Create final .COM file)
                :
                         The actual layout of the diskette is as follows:
                2
                .
                  Track O
                :
                                                Sector
                          1
                                 2
                                       з
                                             4
                                                           6
                                                                 7
                                                                        8
                                                                              9
                ;
                                                    5
                  Head
                                    --+----+
                ;
                                                  -+----
                                                        Boot (<====== CCP ======> (<====== BDOS =======;
                   0
                ;
                                   -+---
                                                                          --+----+
                :
                   1
                         !===== BDOS ====>;<======= BIOS ======>;
                ;
                                     12 13 14 15 16 17
Sector
                         +---+
                           10
                                 11
                                                                             18
                .
                        Equates for defining memory size and the base address and
                         length of the system components
0040 =
                                 FOU
                Memory$Size
                                          64
                                                  ;Number of Kbytes of RAM
                        The BIOS Length must match that declared in the BIOS.
0900 =
                BIOS$Length
                                 EQU
                                          0900H
0200 =
                Boot$Length
                                 EQU
                                          512
0800 =
                CCP$Length
                                 EQU
                                          0800H
                                                  ;Constant
0E00 =
                BDOS$Length
                                 EQU
                                          0E00H
                                                  ;Constant
1F00 =
                Length$In$Bytes EQU
                                          CCP$Length + BDOS$Length + BIOS$Length
0780 =
                                          980H - Boot$Length
                                 EQU
                Start$Image
                                                                   ;Address of CP/M image
                                         Length$In$Bytes + Boot$Length
2100 =
                Length$Image
                                 EQU
                :
                ;
```

Figure 7-5. Example PUTCPM

		;	Disk cha	aracteri	stics	
		7 7 7	the flop one sect	py disk or to th	ette so t	he physical characteristics of hat the program can move from updating the track and resetting Y.
0001		; First\$Se			EQU	1
0012 0009		Last\$Sec Last\$Sec			EQU EQU	18
0200		Sector\$S			EQU	512
		; ; ;	Controll	er char	acteristi	65
		7 7 7	multiple	e sector	s in a si re genera	the floppy disk controller can write .ngle command. However, in order ll example it is shown only reading one
0001	=	; Sectors\$	Per\$Writ	e	EQU	1
		;				
		;	Cold boo	ot chara	cteristic	s
0000	=	; Start\$Tr	ack		EQU	0 ;Initial values for CP/M image
0001	=	Start\$Se	etor		EQU EQU	1 ;= " = (Length\$Image + Sector\$Size - 1) / Sector\$Size
0011	=	Sectors‡ ;	010#Wr10	2	EQU	(Lengthaimage + Sector #Size 17 / Sector#Size
0009		BSPRINTS	\$	EQU	2	;Print string terminated by \$
0005	=	BDOS ; ;		EQU	5	;BDOS entry point
[^] 0100			ORG	100H		
0100		Put\$CPM:		10011		
0100	C33F01		JMP	Main\$Co	de	;Enter main code body ;For reasons of clarity, the main ; data structures are shown before the
0000		CR	EQU	ODH		; executable code. ;Carriage return
A 000	=	LF	EQU	OAH		;Line feed
		Signon\$M				
	0D0A507574 0D0A	ŧ	DB DB	CR,LF, ' CR,LF	Put CP/M	on Diskette′
011B	5665727369	7	DB	'Versio	n í	
0123 0125	3031 20		DW DB	Version		
0126	3037		DW	Month		
0128 0129			DB DW	// Day		
012B	2F		DB	111		
012C 012E	3832 0D0A24		DW DB	Year CR,LF,'	\$1	
		;;	Disk c	ontrol t	ables	
0045	_	; Disk\$Cor	++++	FOU	45H	;5 1/4" control byte
0046	=	Command	Block\$5	EQU	46H	;Control table pointer
0043	=	Disk\$Sta ;	atus	EQU	43H	;Completion status
			as work continue used dim the sec	ing stor es. The rectly a tor numb e sector	age and o sector in s the di- er on the	and DMA\$Address can also be used updated as the load process n the command table cannot be sk controller requires it to be e specified head (1 9) rather on track. Hence a separate variable
		;				

Figure 7-5. (Continued)

0132 02	; Command\$Table;	DB 02H	;Command Write
0133 00	Unit:	DB 02H	;Command write ;Unit (drive) number = 0 or 1
0134 00	Head:	DB 0	;Head number = 0 or 1
0135 00	Track:	DB Start\$T	
0136 00	Sector\$on\$head:		;Converted by low-level driver
0137 0002	Byte\$Count:	DW Sector\$	Size * Sectors\$Per\$Write
0139 8007	DMA\$Address:	DW Start\$I	
013B 4300	Next\$Status:	DW Disk\$St	atus ;Pointer to next status block ; if commands are chained
013D 4500	Next\$Control:	DW Disk\$Co	ntrol\$5 ;Pointer to next control byte ; if commands are chained
	Main\$Code:		
013F 310001	LXI	SP, Put\$CPM	Stack grows down below code
0142 110301	LXI	D,Signon\$Message	s ;Sign on
0145 OE09	MVI	C, B\$PRINTS	Print string until \$
0147 CD0500	CALL	BDOS	
014A 213201	LXI	H,Command\$Table	Point the disk controller at
014D 224600	SHLD	Command\$Block\$5	; the command block
0150 OE11	MVI	C,Sectors\$To\$Wr:	ite ;Set sector count
0152 CD7C01	Write\$Loop: CALL	Put\$CPM\$Write	allerian main and others.
0155 OD	DCR	C Putatria	;Write data onto diskette ;Downdate sector count
0156 CA0000	JZ	õ	;Warm boot
0150 212101	LXI	H.Sector	Alladate ender
0159 213101 015C 3E01		A, Sectors \$Per\$W	;Update sector number rite ; by adding on number of sectors
015E 86	ADD	M, Sectorspreraw	; by controller
015F 77	MOV	M.A	; by controller ;Save result
0160 3E13	MVI	A,Last\$Sector\$0	
0162 BE	CMP	M	
0163 C26F01	JNZ	Not\$End\$Track	
0166 3601	MVI	M EinciteContort	Dn\$Track ;Yes, reset to beginning
0168 2A3501	LHLD	Track	Update track number
016B 23	INX	H	, opbate track indiaber
016C 223501	SHLD	Track	
•	Not\$End\$Track:		
016F 2A3901	LHLD	DMA\$Address	;Update DMA address
0172 110002	LXI	D,Sector\$Size *	Sectors\$Per\$Write
0175 19	DAD	D	
0176 223901	SHLD	DMA\$Address	
0179 C35201	JMP	Write\$Loop	;Write next block
	Put\$CPM\$Write:		At this point, the description of the
			; operation required is in the variables
			<pre>; contained in the command table, along ; with the sector variable.</pre>
0170 05	PUSH	в	Save sector count in C
		-	• • • • • • • • • • • • • •
			atch the disk controller in use
017D 0600	MVI	B,0	;Assume head O
017F 3A3101	LDA	Sector	;Get requested sector
0182 4F	MOV	C,A	;Take a copy of it
0183 FEOA 0185 DA8C01	CPI		lead\$0+1 ;Check if on head 1
0188 D609	JC SUI	Head\$0 Last\$Sector\$on\$H	;No lead\$0 ;Bias down for head 1
018A 4F	MOV	C,A	;Save copy
018B 04	INR	B	;Set head 1
	Head\$0:	-	,
018C 78	MOV	А, В	;Get head
018D 323401	STA	Head	.
0190 79	MOV	A,C	;Get sector
0191 323601	STA	Sector\$On\$Head	

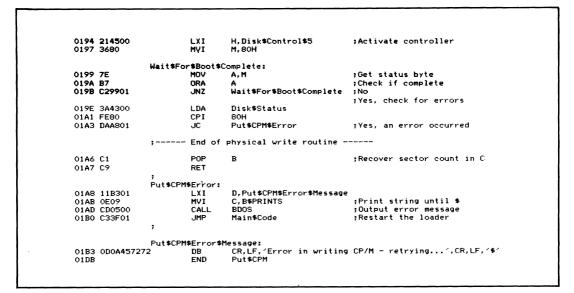


Figure 7-5. (Continued)

Using DDT to Build the CP/M Memory Image

DDT, the Digital Research debug program, is used to read files of type ".COM" and ".HEX" into memory. Understanding the internal structure of these file types is important, both to understand what DDT can do and to understand how the MOVCPM utility can effectively change a machine code file so that it can be executed at a new address in memory.

".COM" File Structure

A COM file is a memory image. It is a replica of the bit patterns that are to be created when the file is loaded into memory. COM files are normally designed to load at location 100H upwards. No internal structure to the file requires this, however, so if you know what the contents of a COM file are, there is nothing to preclude you from loading it into memory starting at some address other than 100H.

As you may recall from the description of the CCP in Chapter 4, the SAVE command built into the CCP allows you to create a COM file by specifying the number of 256-byte "pages" of memory and the name of the file. The CCP will write out an exact image of memory from location 100H up.

".HEX" File Structure

HEX files are output by the assembler. They contain an ASCII character representation of hexadecimal values. For example, the contents of a single byte of memory with the binary value 10101111 would be represented by two ASCII characters, A F, in a HEX file.

The HEX file has a higher level structure than just a series of ASCII characters however. Each line of ASCII characters is terminated by CARRIAGE RETURN/LINE FEED. The overall structure is shown in Figure 7-6.

The most important aspect of a HEX file is that each line contains the address at which the data bytes are loaded. Each line is processed independently, so the load addresses of succeeding lines need not be in order.

DDT can read in a HEX file at an address different from the address where the code must be in order to execute. For example, you can read in the HEX file of the BIOS at the correct place for the memory image (shown in Figure 7-4). There are two ways of using DDT to read in a COM or HEX file. You can specify the name of the file on the same command line with DDT. For example:

		Call up DDT with file name DDT signs on
NEXT PC		
0180 0100	(-	and displays next free byte and entry point address
•	:	and prompts for a commmand

The advantage of this method of loading a file is that you can specify which logical disk is to be searched for the file. The second way of using DDT is to load DDT first, and then, when it has given its prompt, specify the file name and request that DDT load it like this:

- <u>Ifilename.typ<cr></cr></u>	<-	Enter	the	fi	le	name	and	type
- <u>R<er></er></u>	<-	Read :	in t	he	fi	le		

The "I" command initializes the default file control block in the base page (at location 005CH) with the file name and type; it does *not* set up the logical disk. If you need to do this, you must set the first byte of the default FCB manually like this:

- <u>Ifilename.typ<cr></cr></u>	<- Specify file name
- <u>S5C<cr></cr></u>	<- "S"et location 5C
005C 00 <u>02<er></er></u>	<- Was 00, you enter 02 <cr></cr>
005D 41 . <cr></cr>	<- Enter "." to terminate
- <u>R<cr></cr></u>	<- Read in the file

Location 005CH should be set to 01H for Drive A, 02H for B, and so on. The "R" command will read in HEX files to the *execution* addresses specified in each line of the HEX file, so be careful—if you forget to put an ORG (origin)

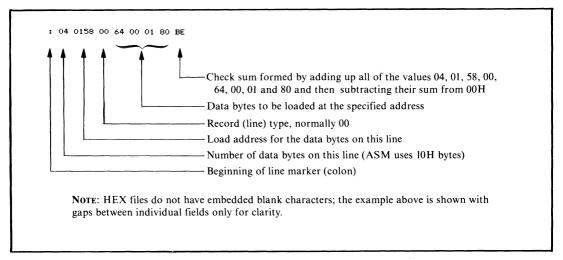


Figure 7-6. Example line from HEX file

statement at the front of the assembly language source code, reading in the resultant HEX file will overwrite location 0000H on up, destroying the contents of the base page. Similarly, if you were trying to read in the HEX file for a BIOS, there is an excellent chance that you will overwrite the currently executing CP/M system.

DDT reacts to the file type you enter as part of the file name. For file types other than .HEX, DDT loads the file starting at location 0100H on up.

The "R" command can also be used to read files into memory at different addresses. You do this by typing a hexadecimal number immediately after the R, with no intervening punctuation. For HEX files, the number that you enter is added to the address in each line of the HEX file and the sum is used as the address into which the data bytes are loaded. The data bytes themselves are not changed, just the load address.

For COM files, the number that you enter is added to 0100H and the sum is used as the starting address for loading the file.

The sum is performed as 16-bit, unsigned arithmetic with any carry ignored, so you can load a BIOS HEX file into low memory by using the "R" command with what is called an "offset value."

If a HEX file has been assembled to execute at address "exec," and you need to use DDT to read in this file to address "load," you need to solve the following equation:

offset = load - exec.

DDT's "H" command performs hexadecimal arithmetic. It calculates and displays the sum of and difference between two hexadecimal values. For example,

the BIOS in Figure 6-4 has been assembled to *execute* at location 0F600H, but needs to be *loaded* into memory at location 1F80H. Here is how to compute the correct offset for the "R" command:

- <u>H1F80,F600<cr></cr></u>	<-	Use	the	н	command
1580,2980	<-	Sum,	di	ff	erence

Thus, to read in the BIOS HEX file called FIG6-4.HEX at location 1F80H, you would enter the following commands to DDT:

- <u>IFIG6-4.HEX<cr></cr></u>	<-	Specify	file name	and ty	ype
- <u>R2980<cr></cr></u>	<-	Load at	0F600H +	2980H ((= 1F80H)

In this way, using DDT, you can read in the HEX files for both the BIOS and the bootstrap loader.

The CP/M Bootstrap Loader

The bootstrap loader is brought into memory by PROM-based firmware in the computer system. It loads in the CCP, BDOS, and BIOS and then transfers control to the cold boot entry point in the BIOS—the first jump instruction in the BIOS jump vector.

The bootstrap loader is a stand-alone program; it cannot make use of any CP/M functions because no part of CP/M is in memory when the bootstrap loader is needed. The firmware in the PROM that loaded the bootstrap may contain some subroutines that can be used by the bootstrap, but this will vary from system to system.

Figure 7-7 shows the bootstrap code for the example BIOS (from Figure 6-4). This code has been written in a general way, so that you can adapt it to your system. The disk controller on the example system can in fact read in multiple sectors from the disk, but for generality the code shown reads in only one sector at a time. This considerably increases the time it takes to load CP/M, but does make the bootstrap loader more general.

Note that almost the first thing that the bootstrap does is to output to the console a sign-on message. Not only does this confirm the version number, but it shows that the bootstrap has been successfully loaded.

The PROM-based code has been designed to load the CP/M bootstrap into location 100H, allowing the code to be debugged as though it were a normal transient program, albeit with minor changes to the address at which it loads the CP/M image from disk. Clearly, this feature is not very helpful if CP/M is being brought up for the first time on a computer system. It helps a great deal, however, if you need to modify the bootstrap or add the capability to boot your system from a new type of disk drive.

Example CP/M cold bootstrap loader ; : This program is written out to track 0, head 0, sector 1 by the PUTCPMF5 program. It is loaded into memory at location 100H on up by the ; . : PROM-based bootstrap mechanism that gets control of the ; CPU on power up or system reset. ; (01/ (07/ 3130 = Version EQU ;Equates used in the sign-on message 3730 = Month FOU FOL 1241 3432 =Day ′82′ 3238 =Year EQU 0000 = ;Set nonzero to debug as normal Debug EQU 0 ; transient program : The actual layout of the diskette is as follows : . : Track 0 Sector ; Δ 1 2 3 5 6 7 8 0 Head _____ Boot : <====== CCP ======>! <====== BDOS =======: 0 ; --+---: ; 1 0 11 12 13 14 1 --+----+---+-; +-10 15 16 17 18 ٠ Sector 7 Equates for defining memory size and the base address and length of the system components. 0040 = Memory\$Size EQU 64 ;Number of Kbytes of RAM The BIOS Length must match that declared in the BIOS. BIOS\$Length 0900H 0900 =EQU 0800Н 0800 = CCP\$Length EQU ;Constant BDOS\$Length OEOOH 0E00 = EQU :Constant ((CCP\$Length + BDOS\$Length + BIOS\$Length) / 1024) + 1 CCP\$Length + BDOS\$Length + BIOS\$Length 0008 = . Length\$In\$K EQU 1F00 =Length\$In\$Bytes EQU : NOT Debug TF CCP\$Entry (Memory\$Size - Length\$In\$K) * 1024 F000 =EQU ENDIF Debug IF CCP\$Entry EQU 3980H ;Read into a lower address. ;This address is chosen to be above ; the area into which DDT initially loads and the 980H makes the addresses similar ; to the SYSGEN values so that the memory image can be checked with DDT. ; ENDIE CCP\$Entry + CCP\$Length + 6 CCP\$Entry + CCP\$Length + BDOS\$Length EQU BDOS\$Entry F806 =BIOS\$Entry EQU F600 =; : **Disk characteristics** ; ; These equates describe the physical characteristics of the floppy diskette so that the program can move from one sector to the next, updating the track and resetting the sector when necessary. 0001 = First\$Sector\$on\$Track EQU 1 18 0012 = Last\$Sector\$on\$Track EQU 0009 = Last\$Sector\$on\$Head\$0 EQU 9 Sector\$Size 0200 = EQU 512 ; Controller characteristics ;

Figure 7-7. Example CP/M cold bootstrap loader

On this computer system, the floppy disk controller can read ; multiple sectors in a single command. However, in order to 2 produce a more general example it is shown only reading one sector at a time. 0001 =Sectors\$Per\$Read EQU 1 Cold hoot characteristics 0000 =Start\$Track FOIL ō ;Initial values for CP/M image
:= " = 0002 =EQU Start\$Sector 2 0010 = Sectors\$To\$Read EQU (Length\$In\$Bytes + Sector\$Size - 1) / Sector\$Size 2 0100 ORG 100H Cold\$Boot\$Loader: 0100 C34001 JMP Main\$Code ;Enter main code body ;For reasons of clarity, the main ; data structures are shown before the executable code. 0000 =CR FOU ODH ;Carriage return LF 000A =EQU OAH ;Line feed 2 Signon\$Message: 0103 0D0A43502F CR, LF, 'CP/M Bootstrap Loader' DB IF Debug DB (Debug) 1 ENDIF 011A ODOA CR,LF DB 011C 5665727369 0124 3031 DB 'Version ' DW Version 0126 20 0127 3037 0129 2F DB DW Month DB 1. 012A 3234 DW Dav 012C 2F 012D 3832 DB 1 Year DW 012F 0D0A00 CR.LF.O DB ; : Disk Control Tables 0045 = Disk\$Control\$5 EQU 45H ;5 1/4" control byte Command\$Block\$5 EQU ;Control table pointer 0046 =**4**6H 0043 = Disk\$Status FOIL 43H ;Completion status The command table track and DMA\$Address can also be used 2 as working storage and updated as the load process continues. The sector in the command table cannot be used directly as the disk controller requires it to be the sector number on the specified head (1 - 9) rather 2 than the sector number on track. Hence a separate variable must be used. 0132 02 Sector: DR Start\$Sector 0133 01 Command\$Table: 01H :Command -- read **DB** ;Unit (drive) number = 0 or 1 ;Head number = 0 or 1 0134 00 Unit: DB 0 0135 00 DB 0 Head: 0136 00 rack ;Used as working variable ;Converted by low-level driver Track: DB Start\$Track 0137 00 Sector\$on\$head: DB ō 0138 0002 Byte\$Count: DW Sector\$Size * Sectors\$Per\$Read 013A 00E0 013C 4300 DMA\$Address: DW CCP\$Entry Next\$Status: DW Disk\$Status ;Pointer to next status block if commands are chained. 2 013E 4500 Next\$Control: DΜ Disk\$Control\$5 ;Pointer to next control byte ; if commands are chained. Main\$Code: 0140 310001 1 X T SP. Cold\$Boot\$Loader :Stack grows down below code



0143 210301 LXI H,Signon\$Message ;Sign on 0146 CDD901 CALL Display\$Message 0149 213301 014C 224600 LXI H,Command\$Table ;Point the disk controller at SHLD Command\$Block\$5 ; the command block 014F 0E10 MVI C.Sectors\$To\$Read :Set sector count Load\$Loop: 0151 CD7B01 0154 OD CALL Cold\$Boot\$Read ;Read data into memory DCR ;Downdate sector count IF NOT Debug 0155 CA00F6 BIOS\$Entry Enter BIOS when load done JZ ENDIF IF Debug JZ ٥ :Warm boot ENDIE 0158 213201 0158 3E01 1 X T H. Sector ;Update sector number ; by adding on number of sectors ; by controller MVI A, Sectors\$Per\$Read 015D 86 ADD M 015E 77 015F 3E13 M, A ;Save result MOV MVI A,Last\$Sector\$On\$Track + 1 ;Check if at end of track 0161 BE CMP 0162 C26E01 JNZ Not\$End\$Track 0165 3601 0167 2A3601 016A 23 016B 223601 MVI M,First\$Sector\$On\$Track ;Yes, reset to beginning LHLD Track :Update track number INX н SHLD Track Not\$End\$Track: 016E 2A3A01 LHLD DMA\$Address ;Update DMA Address 0171 110002 0174 19 0175 223A01 0178 C35101 LXI D,Sector\$Size * Sectors\$Per\$Read DAD n SHLD DMA\$Address IMP Load\$Loop :Read next block Cold\$Boot\$Read: ;At this point, the description of the operation required is in the variables contained in the command table, along with the sector variable. : 017B C5 PUSH в :Save sector count in C ;----- Change this routine to match the disk controller in use -----017C 0600 MVI в, о ;Assume head O 017E 3A3201 LDA Sector ;Get requested sector 0181 4F MOV C,A ;Take a copy of it 0182 FE0A CPI Last\$Sector\$on\$Head\$0+1 ;Check if on head 1 ;No 0184 DA8B01 0187 D609 JC. Head\$0 SUI Last\$Sector\$on\$Head\$0 :Bias down for head 1 0189 4F C,A MOV :Save copy 018A 04 INR ;Set head 1 в Head\$0; 018B 78 MOV ;Get head A.B 018C 323501 018F 79 STA Head MOV A,C ;Get sector 0190 323701 Sector\$On\$Head STA 0193 214500 H,Disk\$Control\$5 ;Activate controller LXI 0196 3680 MVI M.SOH Wait\$For\$Boot\$Complete: 0198 7E :Get status byte MOV A,M 0199 B7 ;Check if complete ORA Δ 019A C29801 Wait\$For\$Boot\$Complete JNZ :No ;Yes, check for errors 019D 3A4300 LDA Disk\$Status 01A0 FE80 CPI 80H 01A2 DAA701 JC Cold\$Boot\$Error ;Yes, an error occurred ;----- End of physical read routine -----

Figure 7-7. (Continued)

01A5 C1 POP в :Recover sector count in C 01A6 C9 RET Cold\$Boot\$Error: 01A7 21B001 LXI H,Cold\$Boot\$Error\$Message 01AA C00901 CALL Display\$Message ;Output error message JMP Main\$Code 01AD C34001 ;Restart the loader 7 Cold\$Boot\$Error\$Message: 01B0 0D0A426F6F DB CR,LF, 'Bootstrap Loader Error - retrying...', CR, LF, 0 Equates for Terminal Output . 0001 = Terminal\$Status\$Port FOU 01H 0002 =Terminal\$Data\$Port EQU 02H 0001 = Terminal\$Output\$Ready FOU 0000\$0001B Display#Message: ;Displays the specified message on the console. ;On entry, HL points to a stream of bytes to be ;output. A OOH-byte terminates the message. ;Get next message byte ;Check if terminator 01D9 7E MOV A.M 01DA B7 ORA Α 01DB C8 RZ ;Yes, return to caller 01DC 4F MOV C. A Prepare for output Output\$Not\$Ready: OIDD DB01 Terminal\$Status\$Port ;Check if ready for output IN 01DF E601 ANI Terminal\$Output\$Ready 01E1 CADDO1 JZ Output\$Not\$Ready ;No, wait 01E4 79 MOV ;Get data character A.C 01E5 D302 OUT Terminal\$Data\$Port ;Output to screen 01E7 23 INX :Move to next byte of message 01E8 C3D901 JMF Display\$Message ;Loop until complete message output :The PROM-based bootstrap loader checks to see that the characters "CP/M" ; are on the diskette bootstrap sector ; before it transfers control to it. ORG 2E0H 02E0 CP/M' 02E0 43502F4D DB 02E4 END Cold\$Boot\$Loader

Figure 7-7. (Continued)

In this case, the bootstrap code must be loaded at location 0780H, not the normal 0980H, because the bootstrap takes a complete 512-byte sector (200H). The same principle applies in determining the offset value to be used with DDT's "R" command to read the bootstrap HEX file, namely:

offset = load address - execution address.

In this case, the values are the following:

0680H = 0780H - 0100H

Using MOVCPM to Relocate the CCP and BDOS

MOVCPM builds a CP/M memory image at the correct locations for SYSGEN, but with the instructions modified to execute at a specific address. Inside MOVCPM is not only a complete replica of CP/M, but also enough

202 The CP/M Programmer's Handbook

information to tell MOVCPM which bytes of which instructions need be changed whenever the execution address of the image needs to be moved.

MOVCPM, as released from Digital Research, contains the bootstrap and BIOS for an Intel MDS-800 computer along with the generic CCP and BDOS. Unless you have an MDS-800, all you use is the CCP and BDOS. Some manufacturers have customized MOVCPM to include the correct bootstrap and BIOS for their own computers; consult their documentation to see if this applies to your computer system.

When you invoke MOVCPM, you have the following options:

MOVCPM<cr>

MOVCPM will relocate its built-in copy of CP/M to the top of available memory and will then transfer control to this new image of CP/M. Unless your manufacturer has included the correct BIOS into MOVCPM, using this option will cause an immediate system crash.

MOVCPM nn<cr>

This is similar to the option above, except that MOVCPM assumes that *nn*K bytes of memory are available and will relocate the CP/M image to the top of that before transferring control. Again, this will crash the system unless the correct BIOS has been installed into MOVCPM.

MOVCPM * *<cr>

MOVCPM will adjust all of the internal addresses inside the CP/M image so that the image could execute at the top of available memory, but instead of actually putting this image at the top of memory, MOVCPM will leave it in low memory at the correct place for SYSGEN to write it onto a disk. The SAVE command could also preserve the image on a disk.

· MOVCPM nn *<cr>

MOVCPM proceeds as above for the "* *" option except that the CP/M image is modified to execute at the top of *nn*K.

MOVCPM has a fundamental problem. The *nn* value indicates that the top of available memory is computed, assuming that your BIOS is small—less that 890 (380H) bytes. If your BIOS is larger (as is the case with the example in Figure 6-4), then you will have to reduce the value of "*nn*" artificially.

Figure 7-8 shows the relationship between the size of the BIOS and the "nn" value to use with MOVCPM. It also shows, for different lengths of BIOS, the BIOS base address, the offset value to be used in DDT to read in the BIOS to location 1F80H (preparatory to using SYSGEN or PUTCPM to write it out), and also the base addresses for the CCP and the BDOS. The base address of the BDOS indicates how much memory is available for loading transient programs, as the CCP can be overwritten if necessary.

The numbers in Figure 7-8 are based on the assumption that you have 64K of memory in your computer system. If this is not the case, then proceed as follows:

- 1. Convert the amount of memory in your system to hex. Remember that 1K is 1024 bytes.
- 2. Determine the length of your BIOS in hex.
- 3. Locate the line in Figure 7-8 that shows a BIOS length equal to or greater than the length of your BIOS.
- 4. Using the "H" command in DDT, compute the BIOS Base Address using the formula:

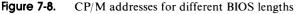
Memory in system - BIOS length from Figure 7-8

5. Find the line in Figure 7-8 that shows the same BIOS Base Address as the result of the computation above. Use this line to derive the other relevant numbers.

It is helpful to use DDT to examine a CP/M image in memory to check that all of the components are correctly placed, and, in the case of the CCP and BDOS, correctly relocated.

Figure 7-9 shows an example console dialog in which DDT is used first to examine the memory image produced by MOVCPM and second to examine the image built into the PUTCPMF utility shown in Figure 7-5.

BIOS	BIOS	DDT	MOVCPM	ССР	BDOS
length	Base	Offset	ʻnn'	Base	Base
600	FA00	2580	64	E400	EC00
A00	F600	2980	63	E000	E800
E00	F200	2080	62	DCOO	E400
1200	EE00	3180	61	D800	E000
1600	EAOO	3580	60	D400	DCOO
1A00	E600	3980	59	DOOO	D800
1E00	E200	3D80	58	CC00	D400
2200	DEOO	4180	57	C800	D000
2600	DAOO	4580	56	C400	CC00
2A00	D600	4980	55	C000	C800
2E00	D200	4D80	54	BCOO	C400
3200	CEOO	5180	53	B800	C000
3600	CAOO	5580	52	B400	BCOO
3A00	C600	5980	51	BOOO	B800
3E00	C200	5080	50	ACOO	B400
\$200	BEOO	6180	49	A800	BOOO
4600	BAOO	6580	48	A400	ACOO
1A00	B600	6980	47	A000	A800
4E00	B200	6D80	46	9000	A400
5200	AEOO	7180	45	9800	A000
5600	AA00	7580	44	9400	9000
5A00	A600	7980	43	9000	9800
5E00	A200	7080	42	8000	9400
5200	9E00	8180	41	8800	9000
5600	9400	8580	40	8400	8000
5 A 00	9600	8980	39	8000	8800
	Apart from the	MOVCPM 'nn' va	lue all other values are	in hexadecimal	

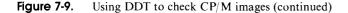


```
Call up MOVCPM requesting a '63K' system
                                       and the image to be left in memory.
A>Moverm 63 *<er>
CONSTRUCTING 63k CP/M vers 2.2
READY FOR "SYSGEN" OR
 "SAVE 34 CPM63.COM"
                                       Save the image from location 100H up. By
                                       convention, the file name is CPMnn.COM, so
                                        in this case it will be CPM63.COM
A>Save 34 cpm63.com<cr>
                                        Call up DDT and request that it read in
                                       CPM63.COM
A>ddt cpm63.com<cr>
DDT VERS 2.2
NEXT PC
2300 0100
                                       Display memory to show the first few bytes of
                                        the CCP. Note the two JMP (C3H) instructions, followed by 7FH, 00H, 20H's, and the Digital
                                        Research Copyright notice. These identify the
                                       research copyright notice. These fuentify the code as being the CCP. Note that the first JMP instruction is to 35CH into the CCP - you can therefore infer the base address of the CCP. In this case the JMP is to locat; on E35C, therefore this version of the CCP has been
                                        configured to execute based at E000H.
-<u>d780,9cf<cr></u>
0980 C3 5C E3 C3 58 E3 7F 00 20 20 20 20 20 20 20 ....X...
C0PYRIGH
 -d980.9cf<cr>
0990 20 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48 COPYRIGH
0940 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49 T (C) 1979, DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 TAL RESEARCH ...
                                       Display the first few bytes of the BDOS. Note
the JMP instruction at 1186. This is the
instruction to which control is transferred
                                       by the JMP in location 5.
-d1180,118F(cr>
1180 00 16 00 00 09 85 C3 11 E8 99 E8 A5 E8 AB E8 B1 .....
                                       Displaying further up in the BDOS identifies
                                        it unambiguously -- there are some ASCII error
                                       messages.
-d1230,126f<cr>
1230 E8 21 DC E8 CD E5 E8 C3 00 00 42 64 6F 73 20 45 .1......Bdos E
1240 72 72 20 4F 6E 20 20 3A 20 24 42 61 64 20 53 65 rr On : $Bad Se
1250 63 74 6F 72 24 53 65 6C 65 63 74 24 46 69 6C 65 ctor$Select$File
1260 20 52 2F 4F 24 E5 CD C9 E9 3A 42 EB C6 41 32 C6 R/D$....B..A2.
                                        Display the first few bytes of the BIOS.
                                       Notice the BIOS JMP vector -- the series of C3H
instructions. Normally the first instruction
                                        in the vector can be used to infer the base
                                        address of the BIOS; in this case it is
                                       F600H. But there is no rule that says that
                                       the cold boot code must be close to the BIOS
JMP vector -- so this is only a rough guide.
-<u>d1f80<cr></u>
1F80 C3 B3 F6 C3 C3 F6 C3 61 F7 C3 64 F7 C3 6A F7 C3 .....a..d..j..
1FB0 C3 B1 F7 82 F6 00 00 00 00 00 00 6E F8 73 F6 0D .....n.s..
1FC0 F9 EE F8 82 F6 00 00 00 00 00 6E F8 73 F6 3C .....n.s.<
1FD0 F9 1D F9 82 F6 00 00 00 00 00 00 6E F8 73 F6 6B .....n.s.k
1FE0 F9 4C F9 82 F6 00 00 00 00 00 6E F8 73 F6 9A .L.....n.s.

      1FE0
      F7
      78
      F9
      A2
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      <
 -^C
```

Figure 7-9. Using DDT to check CP/M images

```
In contrast, load DDT and request that it
                         load the PUTCPMF5.COM program.
A>ddt putcpmf5.com<cr>
DDT VERS 2.2
NEXT PC
2900 0100
                         Display the special bootstrap loader that
                         starts at location 0780H (compared to the
                         MDS-800 bootstrap which is at 0980H). Note
                         the sign-on message.
-<u>d780,7af<cr></u>0780 C3 40 01 0D 0A 43 50 2F 4D 20 42 6F 6F 74 73 74 .@...CP/M Bootst
0770 72 61 70 20 4C 6F 61 64 65 72 0D 0A 56 65 72 73 rap Loader..Vers
07A0 69 6F 6E 20 30 31 20 30 37 2F 32 34 2F 38 32 0D ion 01 07/24/82.
                         Confirm that the CCP is loaded in the correct
                         place. Check the address of the first JMP
                         instruction (OE35CH).
-d980,9bf<cr>
0980 C3 5C E3 C3 58 E3 7F 00 20 20 20 20 20 20 20 20 20 .\..X...
0990 20 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48 C0PYRIGH
09A0 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49 T (C) 1979, DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 TAL RESEARCH ..
                         Confirm that the BDOS is also in place.
-d1180,118f<cr>
1180 00 16 00 00 09 85 C3 11 E8 99 E8 A5 E8 AB E8 B1 .....
                         Confirm that the BIOS has been loaded in the
                         correct place. Check the first JMP to get
some idea of the BIOS base address. Note the
                         sign-on message.
-d1f80<cr>
1F80 C3 F9 F6 C3 0C FE C3 62 F8 C3 78 F8 C3 86 F8 C3 .....b..x....
1FA0 FB C3 41 FB C3 48 FB C3 DE FB C3 F8 FB C3 94 F8 .......
1FB0 C3 B0 FB ED 06 00 00 00 42 6E 25 DF 01 B6 DE 02 ......Bn%.....
1FC0 38 00 00 43 50 2F 4D 20 32 2E 32 2E 30 30 20 30 8..CP/M 2.2.00 0
1FD0 37 2F 31 35 2F 38 32 0D 0A 0A 53 69 6D 70 6C 65 7/15/82...Simple
1FE0 20 42 49 4F 53 0D 0A 0A 44 69 73 6B 20 43 6F 6E BIOS...Disk Con
1FF0 66 69 67 75 72 61 74 69 6F 6E 20 3A 0D 0A 0A 20 figuration :...
2000 20 20 20 20 41 3A 20 30 2E 33 35 20 4D 62 79 74 A: 0.35 Mbyt
2010 65 20 35 22 20 46 6C 6F 70 70 79 0D 0A 20 20 20 e 5" Floppy..
2020 20 04 23 A 20 30 2E 33 35 20 4D 62 79 74 65 20 B: 0.35 Mbyte
2030 35 22 20 46 6C 6F 70 70 79 0D 0A 0A 20 20 20 5" Floppy..
-<u>^C</u>
A>_
```



Putting it all Together

Figure 7-10 shows an annotated console dialog for the complete generation of a new CP/M system. Note that the following file names appear in the dialog:

BIOS1.ASM	Figure	6-4.
PUTCPMF5.ASM	Figure	7-5.
BOOTF5.ASM	Figure	7-7.

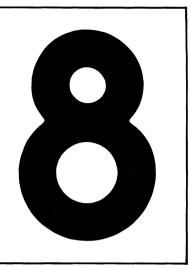
C> <u>asm_bootf5.ccz<cr></cr></u> CP/M_ASSEMBLER - VER 2.0 02E4 004H USE FACTOR END OF ASSEMBLY	Assemble the CP/M Bootstrap Loader, with the source code and HEX file on drive C:, no listing output.
C> <u>asm putcpmf5.ccz≺cr></u> CP/M ASSEMBLER - VER 2.0 01DB	Assemble the PUTCPMF5 program (that writes CP/M onto the disk), with the source code and HEX file on drive C:, no listing output.
003H USE FACTOR END OF ASSEMBLY	
	Assemble the BIOS with the source code and HEX file on drive C:, no listing output.
C> <u>asm biosl.ccz(cr)</u> CP/M ASSEMBLER - VER 2.0 FE6C 011H USE FACTOR END OF ASSEMBLY	
	Start piecing the CP/M image together. Load DDT and ask it to read in the file previously SAVEd after a MOVCPM 63 *.
C> <u>ddt cpm63.com<cr></cr></u> DDT VER\$ 2.2 Next PC 2300 0100	
	Indicate the file name of PUTCPMF5.HEX, and read in without any offset (i.e. it will load at 100H because of the DRG 100H it contains) <u>iputcpmf5.hex<cr></cr></u>
- <u>r≾cr></u> Next PC 2300 0100	
	Indicate the file name of BOOTF5.HEX and read in with an offset of 680H to make it load at 780H on up (it contains ORG 100H too).
- <u>ibootf5.hex<cr></cr></u> - <u>r680<cr></cr></u> NEXT PC 2300 0100	
	Indicate the file name of the BIOS HEX file, and read it in with an offset of 2980 such that it will load at 1F80H (it contains an ORG OF600H).
- <u>ibiosl.hex<cr></cr></u> - <u>r2980<cr></cr></u> NEXT PC 27EC 0000	
- <u>90<cr></cr></u>	Exit from DDT by going to location 0000H and executing a warm boot.
C> <u>save</u> <u>40</u> putcpmf5.com <cr></cr>	Save the complete CP/M image on disk. Saving 40 256-byte pages from location 100H to 2900H.

Figure 7-10. Console dialog for system build

Load and execute the PUTCPMF5 program. C> <u>putcpmf5<cr></cr></u> PUTCPMF5 signs on Put CP/M on Diskette Version 01 07/24/82 and writes the CP/M image to	Program. C> <u>putcpmf5<cr></cr></u> PUTCPMF5 signs on Put CP/M on Diskette Version 01 07/24/82		
Put CP/M on Diskette Version 01 07/24/82 and writes the CP/M image to	Put CP/M on Diskette Version 01 07/24/82 and writes the CP/M image to disk.	C> <u>putcpmf5<cr></cr></u>	
Version 01 07/24/82 and writes the CP/M image to	Version 01 07/24/82 and writes the CP/M image to disk.	Put CP/M on Diskette	PUTCPMF5 signs on

Figure 7-10. Console dialog for system build (continued)

BIOS Enhancements Character Input/Output Data Structures Disk Input/Output Custom Patches to CP/M An Enhanced BIOS



Writing An Enhanced BIOS

This chapter describes ways in which you can enhance your BIOS to make CP/M easier to use, faster, and more versatile.

Get a standard BIOS working on your computer system, and then install the additional features. Although you can write an enhanced BIOS from the outset, it will take considerably longer to get it functioning correctly.

A complete listing of an enhanced BIOS is included at the end of this chapter. It is quite large: approximately 4500 lines of source code, with extensive comments and long variable names to make it more understandable.

The sections that follow describe the main concepts embodied in the enhanced BIOS listing.

BIOS Enhancements

BIOS enhancements fall into two classes: those that add new capabilities and those that extend existing features.

Some enhancements are normally accompanied by utility programs that allow you to select the enhancement option from the console. For example, when the BIOS is enhanced to include a *real time clock*, you need a utility program to set the clock to the correct time. Other enhancements will not require supporting utilities. For example, if the disk drivers are improved to read and write data faster, the enhancement is "transparent." As a user, you are aware of the results of the enhancement but not of the enhancement itself.

Viewed at its simplest, the BIOS deals with two broad classes of input/output:

Character input/output

This includes the console, auxiliary, and list devices.

Disk input/output

This can accommodate several types of floppy and hard disks.

Enhancements in these areas do not fundamentally change the way that the BDOS and CCP interact with these devices. Instead, enhancements improve the way in which the *device drivers* deal with the devices. They can improve the speed of manipulating data, the way of handling external devices, or the user's control over the behavior of the system.

The example enhanced BIOS has capabilities not found in standard CP/M systems. These can be grouped in several main categories:

Character input/output

This area probably benefits most from enhancement. This is partly because such a wide range of peripheral devices needs to be supported and partly because this is the most visible area of interaction between you and your computer. Any improvements here will therefore be immediate and obvious to you as a user.

Error handling

CP/M's error handling is, at best, startling in its simplicity. Enhanced error handling gives you more information about the nature of the failure, and then gives you the options of retrying the operation, ignoring the error, or aborting the program. This topic is covered in detail in Chapter 9.

System date and time

This is the ability to maintain a time-of-day clock and the current date. It allows your programs to set and access the date and time. In addition, your system can react to the passing of time, and you can move certain operations into the time domain. For example, you can set upper limits on the number of seconds, or milliseconds, that each operation should take, and arrange for emergency action if the operation takes too long.

Logical-to-physical device assignment

CP/M's logical-to-physical device assignment is primitive. With enhancements, you can use any character input/output device as the system console, and output data to several devices at the same time.

Disk input/output

CP/M only knows about the 128-byte sector. Even with the deblocking routines shown in Figure 6-4, overall disk performance can be slow. Performance can be improved dramatically by "track buffering" (in which entire tracks are read and written at one time) or by using a *memory disk* (that is, using large areas of RAM as though they were a disk). These have a cost, though, in increased memory requirements.

Public files

CP/M's user number system needs improvements to function well in conjunction with large hard disks.

Preserving User-Settable Options

A by-product of adding features to the BIOS is that many of these features have options that you can alter, either from the console using a utility program or from within one of your programs.

Each of these options, once set according to your preferences, or to the requirements of your hardware, do not normally change from day to day. Therefore, the BIOS should be designed so that options set by the user can be "frozen" or preserved on the disk by using a utility program, FREEZE. All of the variables recording these options are gathered into a single area and then this area is written out to the disk.

This area is called the *configuration block*. In practice, there are two configuration blocks: one short term and the other long term. The short term block is not preservable — you can set options within it, but they cannot be preserved after you switch your computer off. The system date, for example, is normally set each time you turn your computer on, and therefore is kept in the short term block. The baud rate for your printer, on the other hand, is kept in the long term block so that it can be saved permanently.

An extra BIOS entry point, CB\$Get\$Address, has been built into the enhanced BIOS so that utility programs can locate variables in both configuration blocks. For example, when a utility needs to know where the date is kept in memory, it calls CB\$Get\$Address using a code number (specific for date) in a register. CB\$Get\$Address returns the address of the date in memory. If a new version of the BIOS is produced with the date in a different location, CB\$Get\$Address will still hand the correct, although different, address back to the utility program.

212 The CP/M Programmer's Handbook

Two other variables that CB\$Get\$Address can access pertain to the configuration block itself. One is the relative address of the start of the long term configuration block. The other is the length of the long term block. These are used by the FREEZE utility when it needs to preserve the long term block on a disk. FREEZE must (1) read in the sectors containing the long term block from the CP/M BIOS image on the reserved area of the disk, (2) copy the current RAMresident version of the long term block over the disk image version, and then (3) write the sectors back onto the disk.

Figure 8-1 shows how the long term block appears on disk and in memory. The

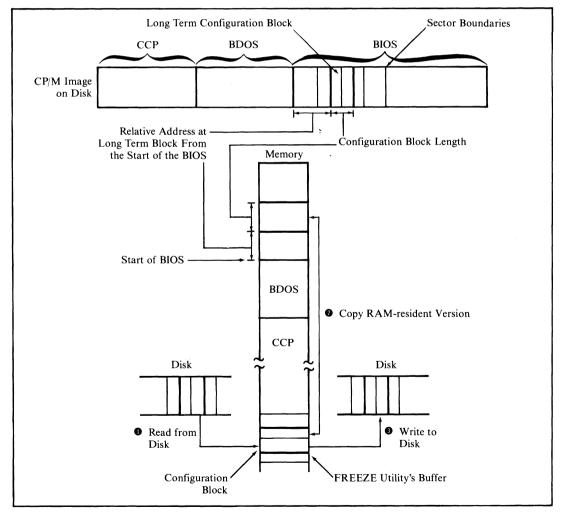


Figure 8-1. Saving the long term configuration block

size of the CCP and BDOS do not change, even if the BIOS does. Therefore, the sector containing the start of the BIOS will not change. The formula (using decimal numbers)

```
BIOS Start Sector + INT(Relative LTB Address / 128)
```

then gives the start sector number to be read in. The number of sectors to read is calculated as follows:

```
(Long Term Block Length + 127)/128
```

The relative address and length can be used to locate the long term block in the BIOS executing in RAM.

Character Input/Output

The character I/O drivers shown in the example BIOS, Figure 8-10, have been enhanced to have the following features:

- A single set of driver subroutines controlling all character devices
- Preservation of option settings
- · Flexible redirection of input/output between logical and physical devices
- · Interrupt-driven input drivers, to get user "type-ahead" capability
- Support of several different protocols to avoid loss of data during highspeed output to printers or other operations
- Forced input of characters into the console input stream, allowing automatic commands at system start-up
- · Conversion of terminal function keys into useful character strings
- Ability to recognize "escape sequences" output to the console and to take special action as a result
- Ability to read the current time and date as though they were typed on the console
- "Timeout" signaling when the printer is busy for too long.

Each of these features is discussed in the following sections, as an introduction to the actual code example.

Single Set of Driver Subroutines

In the following examples, only a single set of subroutines is used to process the input and output for all of the physical devices in the system.

This is made possible by grouping all of the individual device's characteristics

into a table called the *device table*. For example, in order to get a character from the current console device, the address of its device table will be handed over to the subroutines. These in turn will use the appropriate values from the device table when they need to access a port number or any unique attribute of that device.

In our example, the drivers assume that all of the physical devices use serial input/output. To support a device with parallel input/output, you would need to extend the device table to include a field that would enable the drivers to detect whether they were operating on a serial or parallel device. You would probably also have to add different device initialization and input/output routines more suited to the problems of dealing with a parallel port.

The device table structure consists of a series of equate (EQU) instructions. These define the relative offset of each field in the table. Each definition is expressed by referencing the *preceding* field so that you can insert additional fields without revising the definitions for all the other fields.

Individual instances of device tables are then defined as a series of define byte (DB) and define word (DW) lines. The drivers are given the base address of the device table whenever they need to do something with a device. By adding the base address to the relative address (defined by the equate), the drivers can determine the actual address in memory that contains the required value. The detailed contents of the device table are described later in this chapter.

Permanent Setting of Options

About the only options that need preserving in the long term configuration block are the values used to initialize the hardware chips. Other options can be set during automatic execution of the command file when CP/M is first loaded.

Redirection of Input/Output Between Devices

As you recall, the BDOS only "knows about" the *logical* devices console, reader, punch, and list. Using the IOBYTE at location 0003H in conjunction with the STAT utility, you can redirect the BDOS to assign the logical devices to specific physical devices. However, the redirection provided by CP/M is rather primitive. It permits only four physical devices per logical device. Input and output of a logical device must always come from the same physical device. Output data can only be sent to a single destination, or (using the CONTROL-P toggle) to the console and the list device.

The system in Figure 8-10 supports up to 16 physical devices. Any one of these devices can act as the console, reader, punch, or list device. Input can come from any single device. Output can be sent to any or all of the devices. Each logical device's input and output are separate—that is, console input can come from physical device X while the output can be sent to physical devices Y and Z.

Device redirection can be done dynamically, either from within a program or by using a system utility program. For example, if you have some special input device, your program can momentarily switch over to reading input from this device as though it were the console, and then revert back to reading data from the "real" console.

This redirection scheme is achieved by defining a 16-bit word, called the *redirection word*, in the long term configuration block for each of the following logical devices:

- · Console input
- · Console output
- · Auxiliary (reader/punch) input
- · Auxiliary (reader/punch) output
- List input (printers need to send data, too)
- List output.

Each bit in a given redirection word is assigned to a physical device. For input, the drivers use the device corresponding to the first 1 bit that they find in the redirection word. For output, the drivers send the character to be output to all of the devices for which the corresponding bit is set.

The example code does not select a different driver for each bit set — it selects a specific device table and then hands over the base address of this table to the common driver used for all character operations.

Interrupt-Driven Input Drivers

With a standard CP/M BIOS, character data is read from the hardware chips only when control is transferred to the CONIN or READER subroutines. If this character data arrives faster than the BIOS can handle, data overrun occurs and incoming characters are lost.

By using interrupts, the hardware can transfer control to the appropriate interrupt service routine whenever an incoming character arrives. This routine reads the data character and places it into a buffer area to wait for the next CONIN or READER call, which will get the character from the buffer and feed it into the incoming data stream.

User programs and the CCP are "unaware" of this process, perceiving only that data characters are available. However, users will become aware of the process; they will be able to enter data characters from the keyboard before the program is ready for them. This gives the technique its other name—"typeahead." Although this technique does not alter the speed of execution of any programs running under CP/M, it does create the illusion of greater speed, since pauses while a program accepts data vanish completely. The user can enter data at a rate convenient to the tasks or thoughts at hand, without regard to the rate at which the program can accept that data. The example contains the code necessary to handle arriving characters under interrupt control. In order to be of general applicability, the code assumes a "flat" interrupt structure: that is, all character input interrupts cause control to be transferred to the same address in memory. The address is determined by the actual hardware interrupt architecture.

The simplest interrupt schemes use the restart (RST) instructions built into the 8080 CPU chip. In the RST scheme, the external hardware interrupts what the CPU chip is doing and forces one of the eight RST instructions into the processor. Each RST instruction causes the processor to execute what is, in effect, a CALL instruction to a predetermined address in memory.

In more complicated systems, a specific interrupt controller chip (such as the Intel 8259A) will be used. In addition to providing very sophisticated (and complicated) prioritization of interrupts, the interrupt controller can transfer control to a *different* address depending on which physical device causes the interrupt. It does this by forcing the CPU to execute a CALL instruction to a different address for each device.

In both architectures, it is the responsibility of the BIOS writer to initialize all the hardware chips so that an interrupt occurs under the correct circumstances. The BIOS writer also must plant instructions at the correct places in memory to receive control from an RST instruction or from the fake CALL instruction emitted by the interrupt controller.

Some hardware requires that the interrupt service subroutine inform it as soon as the interrupt has been serviced and the character has been input. The example drivers provide for this.

This section deals with using interrupts for the *input* drivers, not the output drivers. All of today's microcomputers can output data much faster than external peripherals can handle. After the first few minutes of output, the computer will fill any reasonably sized buffer — and from this point there is no advantage in having a buffered output system. The computer still must slow down to the peripheral's data rate for each character, although now it is waiting to put the character in the output buffer rather than out to the peripheral.

One exception to this is where you have a large amount of "spare" memory and a "slow" printer (which most of them are). Increasing numbers of systems have more than 64K of RAM. The 8080 or Z80 can't address more than this, but a "bank switched" memory system can switch blocks of memory in and out of that 64K address space.

Using this trick, you can access memory "unknown" to CP/M, store some characters in it, switch back to the normal 64K memory, and return control to the caller of the BIOS output routine. When the physical device is ready to accept another output data character from the CPU, it will generate an interrupt. The interrupt service routine then will access the "secret" buffer, output the characters to the device, and switch back to the normal memory.

For example, if you have a printer that prints at 80 characters per second and

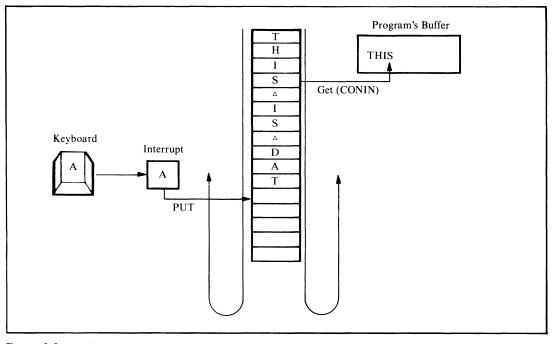


Figure 8-2. Circular buffer type-ahead

you can afford to use 64K of bank switched memory, you can squirrel away 13 minutes of printing—or even more if you design a scheme to compress blanks, storing them in the hidden buffer as a special control sequence.

From the point of view of software, interrupt-driven input drivers are divided into two major groups: the interrupt service routine that reads the characters and stacks them in a buffer, and the non-interrupt routines that get the characters from the buffer and handle the other BIOS functions such as returning console status.

The input character buffer serves as a transfer mechanism between the two groups of subroutines, although the device table also plays an important role.

The example code uses a circular buffer, as shown in Figure 8-2.

The drivers start putting data into the beginning of the buffer. When the last character in the buffer has been reached, the drivers reset to the beginning of the buffer and start over. This, of course, assumes that the non-interrupt drivers have been getting data from the front of the buffer, thus creating space for additional incoming data.

Each device table contains the address of the input buffer, a "put" pointer (for the interrupt service routine), and a "get" pointer (for the non-interrupt service routine). It also contains two character counts: the total number of characters and the number of control characters in the input buffer. You can see how the put and get pointers operate asynchronously. The put pointer is used every time an incoming character generates an interrupt. The get pointer is used for each CONIN call.

The get and put pointers are only single-byte values and are more accurately described as "relative offsets." That is, they contain a value which, when converted to a word and added to the base address of the buffer, will point directly to the appropriate position inside the buffer.

By making the buffer a binary number of characters long -32 characters, for example -a programming trick can be used to make the buffer appear circular. The device tables contain a mask value formed from the buffer's length minus one (length -1). Whenever the get or put pointers are incremented by one (to "point" to the next character position), the updated value is ANDed with this (length -1) mask. In this example, if the get value goes from 31 (the relative address of the last character in the buffer) to 32 (which would be "off the end"), the masking operation will reset it to zero (the relative address of the first character of the buffer). This avoids having to compare pointers to know when to reset them.

It is also simpler to use a count of the number of characters in the buffer, rather than comparing the get and put pointers, to distinguish between an empty and a full buffer. To support different serial protocols, the driver must be able to react when the buffer is within five characters of being full and when it drops below half empty. Both of these conditions are much easier to detect using a simple count that is incremented as a character is put into the buffer and decremented as a character is retrieved from the buffer.

The count of control characters is used to deal with a class of programs that incessantly "gobble" characters, thereby rendering any type-ahead useless. An example is Microsoft's BASIC interpreter. When it is interpreting a program, you can enter a CONTROL-C from the keyboard and the interpreter will come to an orderly stop. It does this by constantly making calls to CONST (console status). If it ever detects an incoming character, it makes a call to CONIN to input the character. A character that is not CONTROL-C is discarded without further ado. Thus, any characters that are input are consumed, destroying the effect of type-ahead.

To deal with this problem, the CONST routine shown in the example can be told to "lie" about the console's status. In this mode, CONST will only indicate that characters are waiting in the input buffer if a control character is received. It uses the control character count to determine whether there are control characters in the buffer; this count is incremented by the interrupt service routine when it detects one, and decremented by the CONIN routine when it gets a control character from the buffer.

Protocol Support

In this context, a protocol is a scheme to avoid loss of data that would otherwise occur if a device sent data faster than the receiving device could handle it. For example, protocols are used to prevent the CPU sending data out to a printer faster than the printer can print the characters and move the paper. The drivers also support input protocols, indicating to a transmitting device when the input buffer gets close to being full.

Two basic methods are used to implement protocols. The first uses the control lines found in the normal RS-232C serial interface cables. For data being output by the computer, the data terminal ready (DTR) signal is used, and for incoming data, the request to send (RTS) signal. These signals conform to the electrical standards for the RS-232C interface; they are considered true when they are at some positive voltage between +3 and +12 volts, and false when they are between -3 and -12 volts.

The second method uses ASCII control characters instead of control signals. Two separate protocols are supported by this method. One uses the ASCII characters XON and XOFF. Before the sending device (the computer or some peripheral device) sends a data character, it checks to see if an XOFF character has been received. If so, the sender will wait for an XON character. The receiving device will only send an XON when it is ready to receive more data.

The second protocol uses the characters ETX (end of transmission) and ACK (acknowledge). This method is normally used only when transmitting data from the computer to a buffered printer. A message length (usually half the printer's buffer size) is defined. When this number of characters has been output, the computer will send an ETX character. No further output will occur until the computer receives an ACK character from the printer.

The example drivers support the DTR high-to-send, the XON/XOFF, and the ETX/ACK protocols for output data. For input, they support RTS high-to-receive and XON/XOFF.

The input protocols are invoked when the input buffer gets within five characters of being full. Then the drivers output an XOFF character or lower the RTS signal voltage, or do both. Only when the input buffer has been emptied to 50% capacity will the drivers send XON or raise the RTS line, or both.

As an emergency measure, if the input buffer becomes completely full, notwithstanding protocols, the drivers will output a predetermined character (defined in the device table) each time they discard an incoming character. This is normally the ASCII BEL (bell) character. When you type too far ahead, the terminal will start beeping to tell you that data is being dropped.

Forced Input into the Console Stream

All application languages provide a means of reading data from the console keyboard. This makes the console input stream a useful gateway to the system. A simple enhancement to the CONIN/CONST routines makes it easy to "fool" the system into acting as if data had been input from the keyboard when in fact the data is coming in from a character string in memory.

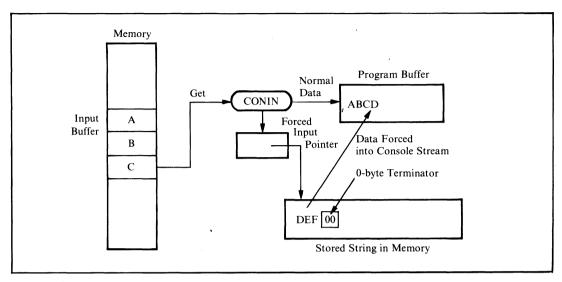


Figure 8-3. CONIN uses forced input data if pointer points to nonzero byte

In the enhanced BIOS, both CONIN and CONST are extended to check a pointer in the long term configuration block, as shown in Figure 8-3.

If this pointer is pointing at a nonzero byte, then that byte is returned as though it had come from the console keyboard. The forced input pointer is then moved up one byte in memory. The process of forcing input continues until a zero byte is encountered.

Forced input serves several purposes. It can be used to force a command or commands into the system when the system first starts up. In conjunction with a utility program, it can allow the user to enter several CP/M commands on a single command line, injecting the characters as each of the commands is executed. It also makes possible the features described in the next two sections.

Support of Terminal Function Keys

Many terminals on the market today have special function keys on their keyboards. When you press one of these keys, the terminal will emit several characters, the first of which is normally the ASCII ESC (escape) character. The remaining one or two characters identify the specific function key that was pressed.

For these function keys to be of any practical use, an applications program must detect the incoming escape sequence and take appropriate action. The problem is that not all terminal manufacturers support the ANSI standard escape sequences. The example drivers avoid this problem by providing a general-purpose method, shown in Figure 8-4, of detecting escape sequences and of substituting a user-defined character string that is injected into the console input stream as though it had been entered from the keyboard.

This scheme permits function keys to be used very flexibly, even for off-theshelf programs that have not been designed specifically to accept function key input.

There is, however, one stumbling block. When an ESCAPE character is received, the progam must detect whether this is the start of a function key sequence or the user pressing the ESCAPE key on the terminal's keyboard. In the former case, the

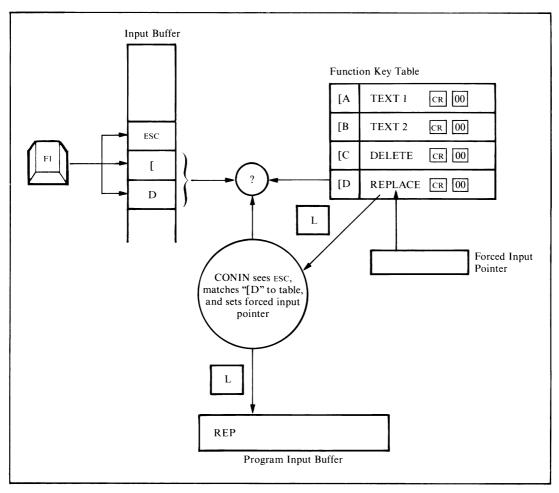


Figure 8-4. CONIN decodes terminal function keys

driver must wait to determine whether a function key string must be substituted for the escape sequence. In the latter case, the driver must input the ESCAPE character as it would other incoming data characters.

This recognition can only be done by moving into the time domain. When the CONIN routine (the non-interrupt routine) gets an ESCAPE character from the input buffer, it delays for approximately 90 milliseconds, enough time for a terminal-generated character sequence to arrive. CONIN then checks the input buffer to see if it contains at least two characters. If it does, the driver checks for a match in a function key table in the long term configuration block. If the characters match a defined function key, then the string associated with the function key will be injected into the console stream by pointing the forced input pointer at it. If the characters do not match anything in the function key table, then the ESCAPE and subsequent characters are handed over as normal data characters.

If after the 90-millisecond delay no further characters have arrived, the ESCAPE character is handed over as a normal character, on the basis that it must have been a manually entered ESCAPE character rather than part of a terminal-generated sequence.

The example drivers show the necessary code and tables for function keys that emit three characters. You could modify them easily for two-character sequences, or, if you are fortunate enough to have a keyboard that uses all eight bits of a byte, to recognize single incoming characters.

Processing Output Escape Sequences

The output side of the console driver, the CONOUT routine, can also be enhanced to recognize escape sequences. It uses a vectored JMP instruction to keep track of the current state of affairs. The CONOUT driver gets an address from the vector and transfers control to it. Normally this vector is set to direct control to the output byte routine. However, if an ESCAPE character is detected in the output stream, the vector is changed to transfer control to a routine that will recognize the character following the ESCAPE. If recognition does not occur, the driver will output an ESCAPE followed by the character that arrived after it.

If the second character is recognized, then the driver can transfer control to the correct escape-sequence processor. This processor can then take whatever action is appropriate. It must also make sure that when all processing is finished, the console output vector is set to process normal output characters again.

This technique is described in more practical detail in the next section, where it is used to preset and read the date and time. You can easily extend the recognition tables in the long term configuration block to perform any special processing that you need, ranging from altering the I/O redirection words to changing any other variable in the system or programming special hardware in your computer.

Be careful not to embed any pure binary values in the sequence of characters going out to the CONOUT routine. If you attempt to send a value of 09H (the TAB

character) out via the BDOS, it will gratuitously expand the tab out to some number of blanks. If you need to send out a bit pattern, such as the I/O redirection word, split it up into a series of 7-bit long values. Then send it out with each byte having the most significant bit set to 1. A value of 09H will then become 89H, preventing the BDOS from expanding it to blanks.

Reading Date and Time From Console

For the moment, set aside the question of how the date and time get into the system. Since the date and time are stored in the short term configuration block (there being no need to save them from one work session to the next), all that the BIOS needs to be able to do is recognize a request from an applications program to read either the date or the time and then set the forced input pointer to the appropriate string in memory. Both the date and time strings are terminated by a LINE FEED followed by a 00 byte.

This sequence of events is shown in Figure 8-5.

You can see that the characters "ESC d" output to CONOUT cause it to point the forced input pointer at the date in memory. Subsequent calls to CONIN bring the characters in the date into the program as though they were being entered on the keyboard.

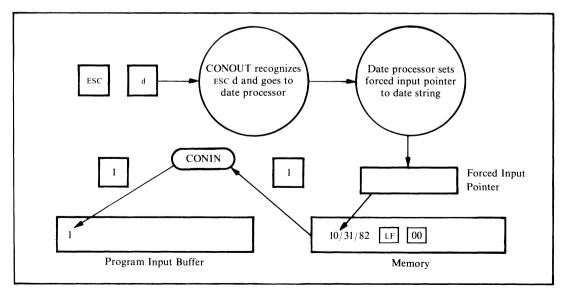


Figure 8-5. Escape sequences sent to CONOUT allow the date to be read by CONIN

"Watchdog" Timeout on Printer

There is no provision in CP/M to deal with a hardware device that for one reason or another is permanently unavailable. Unless special steps are taken in the drivers, the system will screech to a halt in a loop, reading status and testing for the peripheral to be ready.

The example enhancement code shows a scheme, using a real time clock, that can detect when a device such as a printer fails to come ready for more than 30 seconds. On detecting this situation, the code outputs a message to all of the console devices that are not also being used as printers. This type of output is needed to avoid "deadly embraces" where a printer not being ready generates a message that cannot be output because the printer is not ready.

The code that performs the timing function is known as a *watchdog timer*. Each time the real time clock "ticks," the interrupt service routine checks the watchdog count. If the count is nonzero, it is decremented. If the watchdog timer reaches zero, exceeding the time allowed, the drivers will display a message on the console indicating that the printer has been busy for too long. The user then has the option of making the printer ready and trying again to output data, ignoring the error and carrying on, or aborting the program by doing a BDOS System Reset (function 0).

Although sending an error message to the console sounds simple, it is complicated if console output is directed to the offending printer itself. The drivers attempt to solve this problem by sending the message only to those devices being used as consoles and *not* as printers. If all consoles are being used as printer devices as well, the driver will send the message to device 0 — normally the main console.

Keeping Time and Date

CP/M does not have provision for keeping the current time and date in the system. The example enhancement shows how to keep the time of day and the current date in the short term configuration block by using escape sequences output to the console (1) to set them to the correct values and (2) to "read" them from the console input stream.

The example presupposes that the system has a hardware chip that can be programmed to generate an interrupt every 1/60th of a second (16.666 milliseconds). This provides a divide-down counter to measure seconds elapsed. Of course, if your computer has a *true* real time clock that you can read and get the current time in hours, minutes, and seconds, your code will be very simple. You still will need to have the clock generate a periodic interrupt, however, in order to use the watchdog feature for timing printer and disk operations.

Actual time is kept as ASCII characters, using another ASCII control table to determine when "carry and reset to zero" should occur. By changing two bytes in this table, the time can be kept in 12- or 24-hour format.

The date is simply stored as a string. The example code does not attempt to make sure that the date is valid, nor to update when midnight rolls around. This could be done easily by the BIOS — but it would take a fairly large amount of code.

Watchdog Timer

Having a periodic source of interrupts also opens the door to building in an emergency or watchdog timer. This is nothing more than a 16-bit counter. Each time the real time clock interrupts, or ticks, the interrupt service routine checks the watchdog count. If it is already at zero, nothing more happens — the watchdog is not in use. If it is nonzero, the routine decrements the count by one. If this results in a zero value, the interrupt service routine CALLs a predetermined address. This will be the address of some emergency interrupt service routine that can then take special action, such as investigating the cause of the timeout.

The watchdog routine has a non-interrupt-level subroutine associated with it. Calling this set watchdog subroutine provides a means of setting the count to a predetermined number of real time clock "ticks" and setting the address to which control should be transferred if the count reaches zero.

Having called the set watchdog subroutine, the driver can then sit in a status loop, with interrupts enabled, waiting for some event to occur. If the event happens before the watchdog count hits zero, the driver must call the set watchdog routine again to set the count back to zero, thereby disabling the watchdog mechanism.

The watchdog timer can be used to detect printers that are busy for too long or disk drives that take too long to complete an action either because of a hardware failure or because the user has not loaded the disk into the drive.

Data Structures

As already stated, each character I/O device has its own device table that describes all of its unique characteristics.

The other major data structure is the configuration blocks—both short and long term.

This section describes each field in these data structures.

Device Table

Figure 8-6 shows the contents of a device table. More correctly, it shows a series of equates that define the offsets of each field in the device table. The drivers are given the base address of a specific device table. They then access each field by adding the required offset to this base address.

The first part of the device table is devoted to the physical aspect of the device, defining which port numbers are to be used to communicate with it. The drivers need to know several different port numbers since each one is used for a particular

	; The drivers us ; physical devic ; are used to ac ; device table.	ce they s	service. The equ various fields	ates that follow
	;	Port (numbers and state	us bits
0000 = 0001 =	DT\$Status\$Port DT\$Data\$Port	EQU EQU	DT\$Status\$Por	
0002 =	DT\$Output\$Ready	EQU	DT\$DataPort+1	
0003 =	DT\$Input\$Ready	EQU	DT\$Output\$Rea	ut ready status mask / dy+1 t ready status mask
0004 =	DT\$DTR\$Ready	EQU	DT\$Input\$Read	
0005 =	DT\$Reset\$Int\$Port	EQU	DT\$DTR\$Ready+ ;Port	
0006 =	DT\$Reset\$Int\$Value	EQU	DT\$Reset\$Int\$	
0007 =	DT\$Detect\$Error\$Fort	EQU	DT\$Reset\$Int\$	
0008 =	DT\$Detect\$Error\$Value	EQU	DT\$Detect\$Err ;Mask	or\$Port+1 for detecting error (parity etc.)
0009 =	DT\$Reset\$Error\$Port	EQU	DT\$Detect\$Err ;Outp	or\$Value+1 _ ut to port to reset error
000A =	DT\$Reset\$Error\$Value	EQU	DT\$Reset\$Erro ;Valu	e to output to reset error
000B =	DT\$RTS\$Control\$Port	EQU		rol port for lowering RTS
0000 =	DT\$Drop\$RTS\$Value	EQU		e, when output, to drop RTS
000D =	DT\$Raise\$RTS\$Value ;	EQU	DT\$Drop\$RTS\$V ;Valu	alue+1 e, when output, to raise RTS
		e logica	l status (incl.	protocols)
000E =	DT\$Status	EQU	DT\$Raise\$RTS\$	
0001 =	DT\$Output\$Suspend	EQU	0000 \$0001B	;Output suspended pending ; protocol action
0002 =	DT\$Input\$Suspend	EQU	0000\$0010B	;Input suspended until ; buffer empties
0004 =	DT\$Output\$DTR	EQU	0000\$0100B	;Output uses DTR-high-to-send
0008 =	DT\$Output\$Xon	EQU	0000\$1000B	;Output uses Xon/Xoff
0010 =	DT\$Output\$Etx	EQU	0001\$0000B 0010\$0000B	;Output uses Etx/Ack ;Output uses Timeout
0020 =	DT\$Output\$Timeout DT\$Input\$RTS	EQU	0100\$0000B	;Input uses RTS-high-to-receive
0080 =	DT\$Input\$Xon :	EQU	1000\$0000B	;Input uses Xon/Xoff
000F =	DT\$Status\$2	EQU	DT\$Status+1	;Secondary status byte
0001 =	DT\$Fake\$Typeahead	EQU	0000\$0001B	;Requests Input\$Status to ; return "Data Ready" when ; control characters are in ; input buffer
0010 =	; DT\$Etx\$Count	EQU	DT\$Status\$2+1	
0012 =	DT\$Etx\$Message\$Length	EQU	DT\$Etx\$Count+	of chars.sent in Etx protocol 2 tified message length
	•		,	
	;	Input	buffer values	
0014 =	DT\$Buffer\$Base	EQU		ess of input buffer
0016 =	DT\$Put\$Offset	EQU		set for putting chars.into buffer
0017 =	DT\$Get\$Offset	EQU		set for getting chars.from buffer
0018 =	DT\$Buffer\$Length\$Mask	. 200	;Note ; a ;This ; 32 ; 64	<pre>th of buffer - 1 th of buffer length must always be binary number; e.g. 32, 64, or 128, s mask then beccomes: 2 → 31 (0001\$1111B) 4 → 63 (0011\$1111B) 3 → 127 (011\$1111B)</pre>

Figure 8-6. Device table equates

			;After the get/put offset has been
			; incremented it is ANDed with the mask
			; to reset it to zero when the end of
			; the buffer has been reached.
0019 =	DT\$Character\$Count	EQU	DT\$Buffer\$Length\$Mask+1
			;Count of the number of characters
			; currently in the buffer
001A =	DT\$Stop\$Input\$Count	EQU	BT\$Character\$Count+1
			Stop input when the count reaches;
			; this value
001B =	DT\$Resume\$Input\$Count	EQU	DT\$Stop\$Input\$Count+1
			Resume input when the count reaches;
			; this value
001C =	DT\$Control\$Count	EQU	DT\$Resume\$Input\$Count+1
			;Count of the number of control
			; characters in the buffer
001D =	DT#Function#Delay	EQU	DT\$Control\$Count+1
		-	;Number of clock ticks to delay to
			: allow all characters after function
			; key lead-in to arrive
001E =	DT\$Initialize\$Stream	EQU	DT\$Function\$Delay+1
			;Address of byte stream necessary to
			: initialize this device

Figure 8-6. Device table equates (continued)

function. Depending upon your hardware, each port number could be different; however, with standard Intel or Zilog chips, you will often find that the same port number is used for several functions. The drivers also need to know what bit patterns to expect when they read some ports and what values to output to ports in order to obtain particular results.

The layout of the device table and the manner in which the equates are declared are designed to make it easy for you to change the contents of the table to meet your own special requirements. The fields in this first section of the device table are discussed in the sections that follow.

- **DT\$Status\$Port** The driver reads this port to determine whether the hardware chip has incoming data ready to be input to the computer or whether the chip is capable of accepting another data character for output to the physical device.
- **DT\$Data\$Port** The driver reads from this port to access the next data character from the physical device. The driver also writes to this port to output the next data character to the device.

If your computer hardware requires that the input data port be a different number from the output data port, you will have to alter the coding in the device table equates as well as make the necessary changes in the input and output subroutines in the body of the code.

DT\$Output\$Ready This is the bit mask that the driver will AND with the current device status (obtained by reading the DT\$Status\$Port) to see whether the device is ready to accept another output character. It assumes that the device is ready if the result of the AND instruction is nonzero. You may have to change some JNZ (jump

nonzero) instructions to JZ (jump zero) instructions if your hardware device uses inverted logic, with bits in the status byte set to 0 to indicate that the device can accept another character for output.

Note that this status check relates only to the output chip—it is completely separate from the question of whether the peripheral itself is ready to accept data.

- **DT\$Input\$Ready** This is the bit mask that the driver will AND with the current device status to see if there is an incoming data character. The drivers again presume that if the result of the AND is nonzero, then an incoming data character is waiting to be read from the data port. You will need to make changes similar to those for the output subroutines described in the previous section if your hardware uses inverted logic (0 bit means incoming data).
- **DT\$DTR\$Ready** DTR stands for *data terminal ready*. It refers to one of the control lines connected from the actual peripheral device to the I/O chip (via several other integrated circuits). The drivers, as an option, will only output data to the device when the DTR signal is at a positive voltage. If the peripheral, in order to stop the flow of data characters being output to it, lowers the DTR signal to a negative voltage, the drivers will wait. Once DTR goes positive again, the drivers will resume sending data. Many hard-copy devices use this scheme to give themselves a chance to print out data received from the computer. They may have to lower DTR for several seconds, while they perform paper movement, for example.

The value in this field is a bit mask that the drivers use on the device status to determine the state of the data-terminal-ready control signal.

DT\$Reset\$int\$Port Since the input side of the drivers uses interrupts, when an incoming character is ready to be input by the CPU, the hardware generates an interrupt signal, and control is transferred to the interrupt service routine. This routine "services" the interrupt by reading the incoming data character, saving it in memory, and then transferring control back to whatever was being executed when the interrupt occurred.

The more complicated interrupt controller chips (such as the Intel 8259A) must be told as soon as a given interrupt has been serviced so that they can permit servicing of any lower priority interrupts that may be waiting.

This field contains the port number that will be used to "reset" the interrupt, or more correctly, to indicate the end of the previous interrupt's servicing.

- **DT\$Reset\$Int\$Value** This is the value that will be output to the DT\$Reset\$Int\$Port to tell the hardware that the previous interrupt service has been completed.
- **DT\$Detect\$Error\$Port** Before the driver attempts to read any incoming data from the DT\$Data\$Port, it checks to see if any hardware errors have occurred. It does so by reading status from this port.

- **DT\$Detect\$Error\$Value** The status byte that is input from the DT\$Detect\$Error\$Port is ANDed with this value. If the result is nonzero, the driver assumes that an error has occurred.
- **DT\$Reset\$Error\$Port** If an error has occurred, the driver outputs an error reset value to this port number.
- **DT\$Reset\$Error\$Value** This is the value that will be output to the DT\$Reset\$Error\$Port to reset an error.
- **DT\$RTS\$Control\$Port** The drivers use this port number to control the request-to-send line if the RTS protocol option is selected.
- **DT\$Drop\$RTS\$Value** This value is output to the RTS control port to lower the RTS line so that some external device will stop sending data to the computer.
- **DT\$Raise\$RTS\$Value** This value is output to raise the RTS line so that the external device will resume sending data to the computer.
- **DT\$Status** This is the first of two status bytes. It contains bit flags that are set to a 1 bit to indicate the following conditions:

DT\$Output\$Suspend Because of protocol, the device is currently suspended from receiving any further output characters.

DT\$Input\$Suspend

Because of protocol, the device has been requested not to send any more input characters.

DT\$Output\$DTR

The driver will maintain DTR-high-to-send protocol for output data.

DT\$Output\$Xon

The driver will maintain XON/XOFF protocol for output data.

DT\$Output\$Etx

The driver will maintain ETX/ACK protocol for output data.

DT\$Input\$RTS

The driver will maintain RTS-high-to-receive protocol for input data.

DT\$Input\$Xon

The driver will maintain XON/XOFF protocol for input data.

DT\$Status\$2 This is another status byte, also with the following bit flag:

DT\$Fake\$Typeahead

CONST will "lie" about the availability of incoming console characters. It

will only indicate that data is waiting if there are control characters other than CARRIAGE RETURN, LINE FEED, or TAB in the input buffer.

- **DT\$Etx\$Count** This value is only used for ETX/ACK protocol. It is a count of the number of characters sent in the current message. When this count reaches the defined message length, then the driver will send an ETX character and suspend any further output.
- **DT\$Etx\$Message\$Length** This value is the defined message length for the ETX/ACK protocol. It is used to reset the DT\$Etx\$Count,
- DT\$Buffer\$Base This is the address of the first byte of the device's input buffer.
- **DT\$Put\$Offset** This *byte* contains the relative offset indicating where the next incoming character is to be "put" in the input buffer. This byte must then be converted into a word value and added to the DT\$Buffer\$Base address to get the absolute memory location.
- **DT\$Get\$Offset** This byte contains the relative offset indicating where the next character is to be "got" in the input buffer.
- **DT\$Buffer\$Length\$Mask** This byte contains the length of the buffer minus one. The length of the buffer must always be a binary number (8, 16, 32, 64...). Therefore, one less than the length forms a mask value. Both the get and put offsets, after being incremented, are masked with this value. When the offset reaches the end of the buffer, this masking operation will "automatically" reset the offset to zero.
- **DT\$Character\$Count** This is a count of the total number of characters in the buffer. It is incremented by the interrupt service routine each time a character is placed in the buffer, and decremented by the CONIN routine each time it gets a character from the buffer.

CONST uses this value to determine whether any characters are available for sinput.

- **DT\$Stop\$Input\$Count** When the interrupt service routines detect that the DT\$Character\$Count is equal to this value (normally buffer length minus five), the drivers will invoke the selected input protocol, lowering RTS or sending XOFF, to shut off the ' incoming data stream.
- **DT\$Resume\$Input\$Count** When the CONIN routine detects that the DT\$Character\$-Count has become equal to this value, the drivers will again invoke the selected input protocol, either raising RTS or sending XON to resume receiving input data.
- **DT\$Control\$Count** This is a count of the number of control characters in the input buffer. CARRIAGE RETURN, LINE FEED, and TAB characters are not included in this count.

It is incremented by the interrupt service routine and decremented by CONIN. CONST uses the count when the DT\$Fake\$Typeahead mode is active; it will only indicate that characters are waiting in the input buffer if the control count is nonzero.

DT\$Function\$Delay This is the number of clock ticks that should be allowed to elapse after the first character of an incoming escape sequence has been detected. It allows time for the remaining characters in the escape sequence to arrive, assuming that these are being emitted by a terminal at maximum baud rate. Normally, this will correspond to a delay of approximately 90 milliseconds.

DT\$Initialize\$Stream This is the address of the first byte of a string. This string has the following format:

DB	ррН	Port number
DB	nnH	Number of bytes to be output
DB	vvH,vvH	Initialization bytes to be output to the specified port number

This sequence can be repeated as many times as is necessary, with a "port" number of 00H acting as a terminator.

Disk Input/Output

The example drivers show three main disk I/O enhancements:

- Full track buffering
- · Using memory as an ultra-fast disk
- · Improved error handling.

Full Track Buffering

The 5 1/4'' diskettes used in the example system are double-sided. Each side has a separate read/write head in the disk drive. The disk controller is fast enough that, if so commanded, it can read in a complete track's worth of data from one side of the diskette in a single revolution of the diskette.

The drivers have been modified to do just this. The main disk buffer has been dramatically enlarged to accommodate nine 512-byte sectors.

In the earlier standard BIOS, CP/M was configured for tracks of 18 512-byte sectors. The data from each head on a given track was laid "end-to-end" to create the illusion of a single surface with twice as much data on it. For track buffering, performance would be reduced if each read required two revolutions of the diskette, and so in this BIOS the tables and the low-level driver logic have been changed. Each surface is separated, with even numbered tracks on head 0, odd on head 1.

The track number given to the low-level drivers serves two purposes. The least significant bit identifies the head number. When the track number is shifted one bit right, the result is the *physical* track number to which the head assembly must be positioned.

The deblocking algorithm has also been modified by deleting references to sectors. The code is now concerned only with whether the correct disk and track are in the buffer. If this is true, the correct sector must, by definition, be in the buffer.

The deblocking code no longer takes any note when the BDOS indicates that it is writing to an unallocated allocation block—knowledge it used to bypass a sector preread in the standard BIOS. The track size in this enhanced BIOS is much larger than an allocation block, and so the question is meaningless; the whole track must be preread to write just a single sector.

This enhancement really excels when the BDOS is doing directory operations, which always involve a series of sequential reads. The entire directory can be brought into memory, updated, and written back in just two disk revolutions.

One point to watch out for is what is known as "deferred writes." Imagine a program instructed to write on a sector on track 20. The drivers will read in track 20, copy the contents of the designated sector into the track buffer, and return to the program *without* actually writing the data to the disk. The program could "write" to all of the sectors on this track without any actual disk writes. During all this time, this data would exist only in memory and not on the disk drive, so if a power failure occurred, several thousand bytes of data would be lost. Writing to the directory is an exception. The drivers always physically write to the disk when the BDOS indicates that it is writing to a directory sector.

In reality, the increased risk is small. Most programs are constantly reading and writing files, so that the track buffer will be written out frequently in order to read in another track. When programs end, they close output files. This in turn triggers directory writes that force data tracks onto the disk.

If high security is a requirement for your computer, you could extend the watchdog routine to include another separate timer. You could preset this timer for, say, a ten-second delay each time you write into the track buffer but do not write the buffer to the disk. When the count expires, it would set a flag that could be tested by all of the BIOS entry points. If set, they would initiate a write of the track buffer to the disk.

Using Memory as an Ultra-Fast Disk

As you can see from the preceding section, increased performance tends to go hand in hand with increased memory requirements. This is certainly true with a "memory disk," commonly called a RAM-disk or M-disk. In fact, to have an M-disk with reasonable storage capacity, your computer must have at least 128K bytes of additional memory. Since the 8080 or Z80 can only address 64K of memory at one time, to get access to any of this additional memory, some part of your computer's "normal" memory must be removed from the 64K address space and the additional memory must be switched in. This is known as bank-switched memory.

Figure 8-7 shows the memory organization that is supported by the example M-disk drivers.

You can see that the system has a total of 256K bytes of RAM, organized with the top 16K, from 64K down to 48K, being "common"—that is, switched into the address space all the time. The lower 48K can be selected from five banks, numbered 0 to 4. Bank 0 is switched in for normal CP/M operations.

The M-disk parameter blocks describe a disk with eight "tracks," numbered 0 to 7. The least significant bit of the track number determines whether the base address of the track will be 0000H or 6000H. Shifting the track number right one bit gives the bank number. Each track consists of 192 sectors. To get the relative address of a sector within its "track," shift the sector number eight bits left, thus multiplying it by 128.

The M-disk is referenced by logical disk M:. A few special-case instructions are required to return the special M-disk parameter header in SELDSK.

One problem, fortunately easily solved, is that the user's DMA address coexists in the address space with the M-disk image itself. There is no direct way to move data between bank 0 and any other bank. The M-disk uses an intermediary buffer in common memory (above 48K), moving data into this, switching banks, and then moving the data down again. Figure 8-8 shows an example of this sequence, as used when reading from the M-disk.

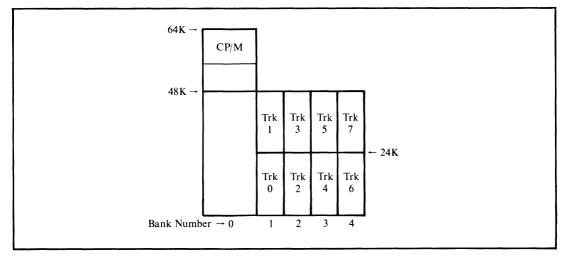


Figure 8-7. Memory organization for M-disk

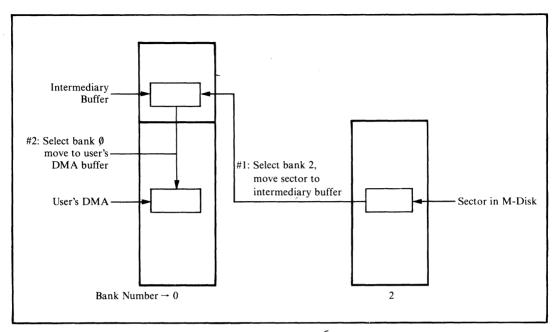


Figure 8-8. Reading a sector from the M-disk image

During cold boot initialization, the M-disk driver checks the very first directory entry (in bank 1) to see if it matches a dummy entry for a file called "M\$Disk." If this entry is present, the M-disk is assumed to contain valid information. If the entry is absent, the initialization code makes this special directory entry and fills the remainder of the directory with 0E5H, making it appear empty. The dummy entry makes it appear that the "M\$Disk" file is in user 15, marked System status and Read-Only—all of which are designed to prevent its accidental erasure.

Custom Patches to CP/M

Two features shown in the enhanced BIOS, one in the CCP and one in the BDOS, require changes to CP/M itself. These features are implemented by modifying the CCP and BDOS to transfer control to the BIOS at specific points, execute a few instructions in the BIOS, and then return to CP/M. The patches could be made by modifying the MOVCPM program to install the changes permanently. The changed version of MOVCPM, however, *must* be used with a specific version of the BIOS. Therefore, patching CP/M "on the fly" ensures that there will be no mismatch between the BIOS and the rest of CP/M.

Both of these patches were produced with the assistance of Digital Research.

User 0 Files Made Public

The first change permits files created in user area 0 to be accessible from all other user numbers. This feature comes into its own only with hard disk systems. On a hard disk, user numbers can partition the disk, but the frequently used utilities must then be duplicated in each user area. Allowing files in user area 0 to be public means that these files will be accessible from all the other user numbers. Hence the files need not be copied into each user area.

The public files feature alters the way that the BDOS performs the Search Next function, allowing access to files declared in user area 0 even when the current user number is not 0. However, the feature is a double-edged sword—user 0 files can be accidentally erased or damaged as well as accessed. Therefore, user 0 files should be declared as System status and Read-Only to protect them. As an additional precaution, public files can be turned off by a control flag in the long term configuration block. This flag is set to an initial state that disables public files.

Modified User Prompt

This modification makes the CCP display the current user number as well as the default disk. For example,

3B>

indicates that you are currently in user number 3, with disk B: as the default. In addition, if you have enabled public files, the prompt is preceded by the letter "P" to serve as a reminder:

P3B>

An Enhanced BIOS

The remainder of this chapter consists of the assembly language source code for the enhanced BIOS described here. It is rather a daunting listing, but will be well worth your study. The copious commentary has been written to make this study easier, and emphasis has been placed on explaining *why* as well as *what* things are done.

As with the standard BIOS, each line is numbered so that you can use the functional index in Figure 8-9 to find areas of interest in the listing. Note that the line numbers are not contiguous. They jump several hundred at the start of each major section or subroutine. This facilitates minor changes in the listing without revision of the functional index. The full listing is given in Figure 8-10.

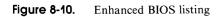
Start Line	Functional Component or Routine
00001	Introductory Comments and Equates
00200	BIOS Jump Table with Additional Private Entries
00400	Long Term Configuration Block
00800	Interrupt Vector
00900	Device Port Numbers and Other Equates
01100	Display\$Message Subroutine
01200	Enter\$CPM Setup
01300	Device Table Equates
01500	Device Table Declarations
01700	General Device Initialization
01800	Specific Device Initialization
02000	Output Byte Stream
02100	CONST Routine
02200	CONIN Routine with Function Key Processing
02500	Console Output
02700	CONOUT Routine with Escape Sequence Processing
02900	AUXIST—Auxiliary Input Status Routine
03000	AUXOST—Auxiliary Output Status Routine
03100	AUXIN—Auxiliary Input Routine
03200	AUXOUT—Auxiliary Output Routine
03300	LISTST—List Status Routine
03400	LIST—List Output Routine
03500	Request User Choice—Request Action After Error
03600	Output Error Message
03656	Get Composite Status from Selected Output Devices
03800	Multiple Output of Byte to All Output Devices
04000	Check Output Device Logically (Protocol) Ready
04200	Process ETX/ACK Protocol
04400	Select Device Table from I/O Redirection Bit Map
04600	Get Input Character from Input Buffer
04800	Introductory Comments for Interrupt-Driven Drivers
04900	Character Interrupt Service Routine Service Device—Puts Character into Input Buffer
05000 05300	Get Address of Character in Input Buffer
05400	Check if Control Character (not CR, LF, TAB)
05500	Output Data Byte
05500	Input Status Routine
05900	Set Watchdog Timer Routine
06000	Real Time Clock Interrupt Service Routine
06200	Shift HL Right One Bit Routine
06300	Introductory Comments for High-Level Disk Drivers
06400	Disk Parameter Headers
06600	Disk Parameter Blocks
06800	SELDSK—Select Disk Routine
07000	SETTRK—Set Track Routine
07100	SETSEC—Set Sector Routine

Figure 8-9. Functional index for listing in Figure 8-10

07200	SETDMA—Set DMA Routine
07300	Skew Tables for Sector Translation
07400	SECTRAN—Sector Translation Routine
07500	HOME—Home Disk to Track and Sector 0
07600	Equates for Physical Disk and Deblocking Variables
07800	READ—Sector Read Routine
07900	WRITE-Sector Write Routine
08000	Common Read/Write Code with Deblocking Algorithm
08300	Move\$8 Routine—Moves Memory in 8-Byte Blocks
08500	Introductory Comments for Disk Controllers
08700	Nondeblocked Read and Write
08900	M-Disk Driver
09100	Select Memory Bank Routine
09200	Physical Read/Write to Deblocked Disks
09400	Disk Error Handling Routines
09700	Disk Control Tables for Warm Boot
09800	WBOOTWarm Boot Routine
10000	Ghost Interrupt Service
10100	Patch CP/M for Public Files and Prompt Changes
10300	Get Configuration Block Addresses
10400	Addresses of Objects in Configuration Blocks
10500	Short Term Configuration Block
10700	Note on Why Uninitialized Buffers are at End of BIOS
10800	Cold Boot Initialization Hidden in Disk Buffer Followed by All Uninitialized Buffers

FIGURE 8-9. Functional index for listing in Figure 8-10 (continued)

	00001	. This i	c a chal	atal ava	role of an enhanced BIOS				
	00010	: This is a skeletal example of an enhanced BIOS. : It includes fragments of the standard BIOS							
	00011	; shown as Figure 6-4 in outline, so as to							
	00012				e enhancements with the				
	00013				e. Many of the original				
	00014				en abbreviated or deleted				
	00015	; entire		indire bei					
	00016								
	00017	:< NOTE:	NOTE: The line numbers at the left are included						
	00018 ; to allow reference to the code from the text.								
	00019		There are deliberate discontinuities in the						
	00020				ow space for expansion.				
	00021								
3030 =	00022	VERSION	EQU	1001	;Equates used in the sign-on message				
3230 =	00023	MONTH	EQU	1021					
3632 =	00024	DAY	EQU	1261					
3338 =	00025	YEAR	EQU	1831					
	00026	;							
	00027	************	********	*********	***********	***			
	00028	;*				×			
	00029	;∗ This I	BIOS is f	or a com	puter system with the following	×			
	00030	;* hardwa	hardware configuration :						
	00031	;*				*			
	00032	;*	808			×			
	00033	;*		bytes o		×			
	00034	;*			O ports (using signetics 2651) for:	*			
	00035	;*			ommunications and list	×			
	00036	;*			mini floppy, double-sided, double-	×			
	00037	;*			ives. These drives use 512-byte sectors.	*			
	00038	;*			used as logical disks A: and B:.	*			
	00039	;*	Fu	ll track	buffering is supported.	*			



	00040	;*	-	Two	8" stand	lard diskette drives (128-byte sectors) *	
	00041	;*				used as logical disks C: and D:. *	
	00042	;*	-			ed disk (M-disk) is supported. *	-
	00043	;*				*	-
	00044	;*				ent disk controllers are used, one for 👘 🕷	
	00045	;*				te type. These controllers access memory *	
	00046	;*				oth to read the details of the *	
	00047	;*				they are to perform and also to read *	
	00048	;*		and	write c	lata from and to the diskettes. *	
	00049 00050	;* 				×	
	00051	• ********	*******	******	*******	× ************************************	
	00052	,	~~~~~~				
	00053						
	00054	; E	quates f	or cha	racters	in the ASCII character set	
	00055	-					
0011 =	00056	XON E	QU 1	11H	:Reenat	les transmission of data	
0013 =	00057			зн		es transmission of data	
0003 =	00058	ETX E	ເຊບ ດ	ээн		transmission	
0006 =	00059	ACK E	ເຊບ ດ	06H	;Acknow	ledge	
000D =	00060			DH	;Carria	ige réturn	
000A =	00061			DAH	;Line f	eed	
0009 =	00062)9H		ontal tab	
0007 =	00063	BELL E	ເຊບ ດ	07H	;Sound	terminal's bell	
	00064	;					
	00065	;		_			
	00066					mory size and the base address and	
	00067	; 1	ength of	the s	ystem co	omponents	
	00068	;				NUL CONTRACTOR	
0040 =	00069 00070	Memory\$Si	ze E	EQU	64	Number of Kbytes of RAM	
	00071	, т	ba PIOS	longth	muct be	determined by inspection.	
	00072					S\$Entry line below by changing the first	
	00072					on (this will make the assembler start	
	00074					. Then assemble the BIOS and round up to	
	00075					dress displayed on the console at the end	
	00076		of the as			diess displayed on the console at the end	
	00077	;		, , ,	•		
2500 =	00078	BIOS\$Leng	th E	EQU	2500H	;< Revised to an approximate value	
	00079					; to reflect enhancements	
	00080	;					
0800 =	00081	CCP\$Lengt		EQU	0800H	;Constant	
0E00 =	00082	BDOS\$Leng	th E	EQU	OEOOH	;Constant	
	00083	;					
000F =	00084	Overall\$L	ength E	EQU	(CCP\$Le	ngth + BDOS\$Length + BIOS\$Length + 1023) /	1024
	00085	;					
C400 =	00086	CCP\$Entry		EQU		\$Size - Overall\$Length) * 1024	
CC06 =	00087	BDOS\$Entr		EQU			
DA00 =	00088	BIOS \$E ntr				ry + CCP\$Length + 6	
		DIOS#ENC	y E	EQU		ry + CCP\$Length + 8 ry + CCP\$Length + BDOS\$Length	
	00089	;			CCP\$Ent	ry + CCP\$Length + BDOS\$Length	
0005 =	00090	; BDOS		EQU EQU		ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making	
0005 =	00090 00091	; BDOS			CCP\$Ent	ry + CCP\$Length + BDOS\$Length	
0005 =	00090 00091 00092	; BDOS ;			CCP\$Ent	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making	
0005 =	00090 00091 00092 00200	; BDOS ; ;#	E	EQU	CCP\$Ent	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests)	
0005 =	00090 00091 00092 00200 00201	; BDOS ; ;# ; 0	E		CCP\$Ent	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making	
0005 =	00090 00091 00092 00200 00201 00202	; BDOS ; ; ; ; ; 0	E RG B	EQU 8105 \$ En	CCP\$Ent 0005H try	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests)	
0005 =	00090 00091 00092 00200 00201 00202 00203	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E	EQU 8105 \$ En	CCP\$Ent 0005H try	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests)	
	00090 00091 00092 00200 00201 00202 00203 00204	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E RG B IOS jump	BIOS\$En vector	CCP\$Ent 0005H try r	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address	
0005 = 0000 C31311	00090 00091 00092 00200 00201 00202 00203 00204 00205	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E RG B IOS jump MP B	EQU 8105 \$ En	CCP\$Ent 0005H try r ;Cold b	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address poot entered from CP/M bootstrap loader	
	00090 00091 00092 00200 00201 00202 00203 00203 00204 00205 00206	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E RG B IOS jump MP B	BIOS\$En vector	CCP\$Ent 0005H try r ;Cold b ; Labe	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address poot entered from CP/M bootstrap loader iled so that the initialization code can	
	00090 00091 00092 00200 00201 00202 00203 00204 00205 00206 00207	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E RG B IOS jump MP B	BIOS\$En vector	CCP\$Ent 0005H try r ;Cold b ; Labe ; Put	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address noot entered from CP/M bootstrap loader lled so that the initialization code can the warm boot entry address in location	
0000 C31311	00090 00091 00092 00200 00201 00202 00203 00204 00205 00206 00207 00208	; BDOS ; ; ; ; ; ; B ; ; B ; ; B ; ; ; ; ; ;	E RG B IOS jump MP B \$Entry:	EQU BIOS\$En • vector BOOT	CCP\$Ent 0005H try ;Cold b ; Labe ; put ; 0001	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader iled so that the initialization code can the warm boot entry address in location H and 0002H of the base page	н
	00090 00091 00092 00200 00201 00202 00203 00204 00205 00206 00207 00208 00209	; BDOS ; ; ; ; ; ; B ; ; B ; ; B ; ; ; ; ; ;	E RG B IOS jump MP B \$Entry:	BIOS\$En vector	CCP\$Ent 0005H try r ;Cold b ; Labe ; Put ; Q001 ;Warm b	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader lled so that the initialization code can the warm boot entry address in location H and 0002H of the base page moot entered by jumping to location 0000	н
0000 C31311	00090 00091 00200 00201 00202 00203 00204 00205 00206 00207 00208 00209 00210	; BDOS ; ; ; ; ; ; B ; ; B ; ; B ; ; ; ; ; ;	E RG B IOS jump MP B \$Entry:	EQU BIOS\$En • vector BOOT	CCP\$Ent 0005H try ;Cold b ; Labe ; Put ; O001 ;Warm b ; Relo	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader iled so that the initialization code can the warm boot entry address in location H and 0002H of the base page moot entered by jumping to location 0000 ads the CCP, which could have been _	н
0000 C31311	00090 00091 00092 00200 00201 00202 00203 00204 00205 00206 00207 00208 00209 00209	; BDOS ; ; ; ; ; ; B ; ; B ; ; B ; ; ; ; ; ;	E RG B IOS jump MP B \$Entry:	EQU BIOS\$En • vector BOOT	CCP\$Ent 0005H try ;Cold b ; Labe ; Put ; 0001 ;Warm b ; Relo ; over	ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address noot entered from CP/M bootstrap loader lled so that the initialization code can the warm boot entry address in location H and OuO2H of the base page noot entered by jumping to location 0000 ads the CCP, which could have been written by previous program in transient	н
0000 C31311 0003 C3750E	00090 00091 00092 00200 00201 00202 00203 00204 00205 00206 00207 00206 00207 00208 00209 00210 00211	; BDOS ; ; ; ; ; B ; B ; U Warm\$Boot	E RG B IOS jump MP B \$Entry: MP W	BIOS\$En 9 vecto 800T 1800T	CCP\$Ent 0005H try ;Cold b ; Labe ; put ; ovor ;Warm b ; Relo ; over ; prog	<pre>ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address oot entered from CP/M bootstrap loader lled so that the initialization code can the warm boot entry address in location H and 0002H of the base page ioot entered by jumping to location 0000 ads the CCP, which could have been written by previous program in transient ram area</pre>	
0000 C31311	00090 00091 00092 00200 00201 00202 00203 00204 00205 00205 00206 00207 00208 00209 00210 00211 00212	; BDOS ; ; ; ; ; B ; B ; U Warm\$Boot	E RG B IOS jump MP B \$Entry: MP W	EQU BIOS\$En • vector BOOT	CCP\$Ent 0005H ;Cold b ; Labe ; put ; O001 ;Warm b ; Relc ; over ; prog ;Consol	<pre>ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader iled so that the initialization code can the warm boot entry address in location H and 0002H of the base page moot entered by jumping to location 0000 ads the CCP, which could have been written by previous program in transient ram area e status returns A = 0FFH if there is a</pre>	
0000 C31311 0003 C3750E 0006 C32B03	00090 00091 00092 00200 00201 00202 00203 00204 00205 00206 00207 00208 00209 00207 00208 00209 00210 00211 00212 00213 00214	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E RG B IOS jump MP B \$Entry: MP W	BIOS\$En vector BOOT BOOT	CCP\$Ent 0005H try ;Cold b ; Labe ; Put ; Oold ;Warm b ; Relo ; over ; prog ;Consol ; consol	<pre>ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address noot entered from CP/M bootstrap loader lled so that the initialization code can the warm boot entry address in location H and 0002H of the base page noot entered by jumping to location 0000 ads the CCP, which could have been written by previous program in transient ram area e status returns A = 0FFH if there is a nole keyboard character waiting</pre>	
0000 C31311 0003 C3750E	00090 00091 00092 00200 00202 00203 00204 00205 00206 00206 00207 00208 00209 00210 00211 00212 00213 00214 00215	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E RG B IOS jump MP B \$Entry: MP W	BIOS\$En 9 vecto 800T 1800T	CCP\$Ent 0005H try ;Cold b ; Labe ; Put ; Rela ; Rela ; Rela ; over ; Prog ;Consol ; Consol ; Consol	<pre>ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader lled so that the initialization code can the warm boot entry address in location H and 0002H of the base page loot entered by jumping to location 0000 mods the CCP, which could have been _ written by previous program in transient ram area e status returns A = 0FFH if there is a lole keyboard character waiting e input returns the next console keyboard</pre>	
0000 C31311 0003 C3750E 0006 C32B03	00090 00091 00092 00200 00201 00202 00203 00204 00205 00206 00207 00208 00209 00207 00208 00209 00210 00211 00212 00213 00214	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E RG B IOS jump MP B HEntry: MP W MP C MP C	BIOS\$En vector BOOT BOOT	CCP\$Ent 0005H ; Cold b ; Labe ; Labe ; Put ; 0001 ; Rela ; over ; Prog ; Consol ; consol ; consol	<pre>ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader iled so that the initialization code can the warm boot entry address in location H and 0002H of the base page moot entered by jumping to location 0000 ads the CCP, which could have been written by previous program in transient ram area e status returns A = 0FFH if there is a mole keyboard character waiting e input returns the next console keyboa acter in A</pre>	
0000 C31311 0003 C3750E 0006 C32D03 0009 C33A03	00090 00091 00092 00200 00201 00202 00203 00204 00205 00206 00207 00208 00207 00208 00207 00208 00207 00210 00211 00212 00213 00214	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E RG B IOS jump MP B HEntry: MP W MP C MP C	EQU BIOS\$En: Vector BOOT VBOOT CONST CONST	CCP\$Ent 0005H try ;Cold b ; Labe ; Put ; Cold b ; Labe ; Put ; Relc ; Over ; Prog ;Consol ; Consol ; Consol	<pre>ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader lled so that the initialization code can the warm boot entry address in location H and 0002H of the base page loot entered by jumping to location 0000 mods the CCP, which could have been _ written by previous program in transient ram area e status returns A = 0FFH if there is a lole keyboard character waiting e input returns the next console keyboard</pre>	
0000 C31311 0003 C3750E 0006 C32D03 0009 C33A03	00090 00091 00092 00200 00202 00202 00203 00204 00205 00206 00207 00208 00207 00208 00207 00210 00211 00212 00213 00214 00215 00216 00217	; BDOS ; ; ; ; ; B ; ; B ; U Warm\$Boot U U U U U	E RG B IOS jump MP B \$Entry: MP W MP C MP C	EQU BIOS\$En: Vector BOOT VBOOT CONST CONST	CCP\$Ent 0005H ;Cold b ; Labe ; put ;Warm b ; Relc ; over ; consol ; consol ; consol ; char	<pre>ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader lled so that the initialization code can the warm boot entry address in location H and 0002H of the base page moot entered by jumping to location 0000 ads the CCP, which could have been _ written by previous program in transient ram area e status returns A = 0FFH if there is a nole keyboard character waiting e input returns the next console keyboa acter in A</pre>	
0000 C31311 0003 C3750E 0006 C32D03 0009 C33A03 0000 C3D703	00090 00091 00092 00200 00201 00202 00203 00204 00205 00205 00206 00207 00208 00207 00208 00207 00210 00211 00212 00213 00214 00215 00216	; BDOS ; ; ; ; ; B ; ; B ; U Warm\$Boot U U U U U	E RG B IOS jump MP B \$Entry: MP W MP C MP C	EQU 310S\$En: 2 vector 300T 4BOOT CONST CONST CONST CONST	CCP\$Ent 0005H try ;Cold b ; Labe ; Put ; Relc ; Otr ; Consol ; Consol ; Consol ; Consol ; the ;List c ; list c	<pre>ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader iled so that the initialization code can the warm boot entry address in location H and 0002H of the base page moot entered by jumping to location 0000 ads the CCP, which could have been _ written by previous program in transient ram area e status returns A = 0FFH if there is a nole keyboard character waiting e input returns the next console keyboa acter in A e output outputs the character in C to console device</pre>	rd /
0000 C31311 0003 C3750E 0006 C32D03 0009 C33A03 0000 C3D703	00090 00091 00092 00200 00201 00202 00203 00204 00205 00206 00207 00208 00207 00208 00207 00208 00207 00211 00212 00213 00214 00215	; BDOS ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	E RG B IOS jumF MP B HEntry: MP W MP C MP C MP L	EQU 310S\$En: 2 vector 300T 4BOOT CONST CONST CONST CONST	CCP\$Ent 0005H try ;Cold b ; Labe ; put ; alt ; over ; prog ;Consol ; consol ; consol ; consol ; the ;List c ; list ;Auxili	<pre>ry + CCP\$Length + BDOS\$Length ;BDOS entry point (used for making ; system reset requests) ;Assemble code at BIOS address moot entered from CP/M bootstrap loader iled so that the initialization code can the warm boot entry address in location H and 0002H of the base page moot entered by jumping to location 0000 ads the CCP, which could have been _ written by previous program in transient ram area e status returns A = 0FFH if there is a mole keyboard character waiting e input returns the next console keyboa acter in A e output outputs the character in C to console device utput outputs the character in C to the</pre>	rd /

Figure 8-10. (Continued)

0015	C3A104	00223	JMP	AUXIN	;Auxiliary input returns the next input character from
		00224			; the logical auxiliary device in A
	C3160A	00225	JMP	HOME	;Homes the currently selected disk to track O
001B	C36309	00226	JMP	SELDSK	;Selects the disk drive specified in register C and
		00227			; returns the address of the disk parameter header
001E	C39B09	00228	JMP	SETTRK	;Sets the track for the next read or write operation
		00229			; from the BC register pair
0021	C3A109	00230	JMP	SETSEC	;Sets the sector for the next read or write operation
		00231			; from the A register
0024	C3A809	00232	JMP	SETDMA	;Sets the direct memory address (disk read/write)
		00233			; address for the next read or write operation
		00234			; from the DE register pair
0027	C3370A	00235	JMP	READ	Reads the previously specified track and sector from
		00236			; the selected disk into the DMA address
002A	C34B0A	00237	JMP	WRITE	;Writes the previously specified track and sector onto
		00238			; the selected disk from the DMA address
0028	C3D704	00239	JMP	LISTST	;Returns A = OFFH if the list device(s) are
		00240	INF	OFOTOWN	; logically ready to accept another output byte
0030	C3100A	00241 00242	JMP	SECTRAN	;Translates a logical sector into a physical one
			, Andibia		vate" BIOS entry points
		00243 00244		nai pri	vate blos entry points
0033	C38F04	00244	; JMP	AUXIST	;Returns $A = OFFH$ if there is input data for
0033	0.00004	00243	Of #	HUNIOI	; the logical auxiliary device
0034	C39B04	00248	JMP	AUXOST	; Returns A = OFFH if the auxiliary device(s) are
0000	00/004	00248	0.1		; logically ready to accept another output byte
0039	C3FA02	00248	JMP	Specifi	c\$CIO\$Initialization
0007	COLHOZ	00250	wi n	CPECI II	;Initializes character device whose device
		00251			; number is in register A on entry
0030	C36D08	00252	JMP	Set\$Wat	
0000		00253	0.1	Se t and t	;Sets up watchdog timer to CALL address specified
		00254			; in HL, after BC clock ticks have elapsed
003E	C33C0F	00255	JMP	CB\$Get\$	
0000	000001	00256	Ci li	0240014	;Configuration block get address
		00257			; Returns address in HL of data element whose
		00258			; code number is specified in C
		00259	;		
		00400	;#		
		00401	; Long te	rm confi	guration block
		00402	;		
		00403	Long\$Term\$CB:		
		00404	;		
		00405	;		
		00406	; Public	files (f	iles in user O accessible from all
		00407	; other u	ser numb	ers) enabled when this flag is set
		00408	; nonzero	•	
		00409	;		
0042	00	00410	CB\$Public\$Files	:	DB O ;Default is OFF
		00411	;		
		00412	;		
		00413		ced inpu	t pointer is initialized to point to the
		00414	· · · · · · · · · · · · · · · · · · ·		g of characters. These are injected into
		00415	; the con	sole inp	ut stream on system start-up.
		00416	;		
0043	5355424D4		CB\$Startup:		DB SUBMIT STARTUP*,LF,0,0,0,0,0,0
		00418	;	A	ingl device redirection
)	00419		to phys	ical device redirection
	7	00420	;	East 1-	cial device bas a light word accorded
		00421	;		gical device has a 16-bit word associated
		00422	;	with it	. Each bit in the word is assigned to a
		00423	;	specifi	c physical device. For input, only one bit
		00424	,		set input will be read from the
		00425 00426	;		onding physical device. Output can be d to several devices, so more than one
		00426	;		be set.
		00427	;	ort can	De Jeli
		00428	;	The fri	lowing equates are used to indicate
		00429	;		c physical devices.
		00430	;	Specifi	a physicae wateries
		00432	;		1111 11)
		00433	:		5432 1098 7654 3210)<- Device number
0001	=	00434	, Device\$0	EQU	0000\$0000\$0000\$00001B
0002		00435	Device\$1	EQU	0000\$0000\$0000\$00010B
0004		00436	Device\$2	EQU	0000\$000\$0000\$0100B
		00437	;		
		00438	;	The fol	lowing words are tested by the logical
		00439	;		drivers to transfer control to

		00440				
		00440 00441	; ti ;	ne appropria	ite physical	device drivers
	0100	00442	CB\$Console\$Input:	DW	Device\$0	
005A	0100	00443 00444	CB\$Console\$Output:	: DW	Device\$0	
0050	0200	00445	, CB\$Auxiliary\$Inpu	t: DW	Device\$1	
005E	0200	00446	CB\$Auxiliary\$Outp	ut: DW	Device\$1	
0040	0400	00 4 47 00 44 8	; CB\$List\$Input;	νnω	Device\$2	
	0400	00449	CB\$List\$Output:	DW	Device\$2	
		00450	7			
		00451 00452				pecific bits in the o specific device
		00452			y the physic	
		00454	1			
	0500	00455	CB\$Device\$Table\$A			
	8E02 AE02	00456 00457		T\$0 T\$1		
	CE02	00458		T\$2		
006A	0000000000			,0,0,0,0,0,0,0	,0,0,0,0,0,0,0	;Unassigned
		00 4 60 00 4 61	;			
		00462	; Device in:	itialization	byte stream	s
		00463	;			· · · · · ·
		00464 00465	; These ini ; initializa	tialization	streams are	output during the device st whenever the baud rate
		00466	; needs to t	be changed.	They are def	st whenever the baud rate ined in the long term
		00467	; configura	tion block s	o as to "fre	eze" their contents from one
		00 4 68 00 4 69	; system sta ;	artup until	the next.	
		00470		ss of each s	tream is con	tained in each device table.
		00471	;			
		00 4 72 00 4 73	; The stream;	m format is:		
		00474	, ; DI	в хх	;	Port number (OOH terminates)
		00475	; DI			Number of bytes to output to port
		00 4 76 00 4 77	; DI	B vv,vv	;,,,;;	Values to be output
		00478	, DO\$Initialize\$Stre	eam:	;Example	data for an 8251A chip
0084		00479		EDH		Port number for 8251A Number of bytes
0085	000000	00480 00481	DB 6 DB 0	,0,0		Dummy bytes to get chip ready
0089		00482		100\$0010B	;	Reset and raise DTR
008A	6E	00483	DB O	1\$10\$11\$10B	;	1 stop, no parity, 8 bits/char, divide down of 16
008B	25	00484 00485	DB O	010\$0101B	,	RTS high, enable Tx/Rx
		00486			;Example	data for an 8253 chip
008C 008D		00487 00488	DB OI DB 1	DFH		Port number for 8253 mode Number of bytes to output
008D		00488		1\$11\$011\$0B		Select:
		00490			;	Counter 1
		00491 00492			;	Load LS byte first Mode 3, binary count
008F	DE	00492	DB OI	DEH	;	Port number for counter
0090		00494	DB 2		;	Number of bytes to output
		00495	DO\$Baud\$Rate\$Const			Label used by utilities 9600 Baud (based on 16x divider)
0091	0700 00	00 4 96 00 4 97	DW OG DB O	007H		9600 Baud (based on 16x divider) Port number of 00 terminates stream
0070		00498				
		00499	D1\$Initialize\$Stre		;Example	data for an 8251A chip Bart number for 8251A
0094		00500 00501	DB 01 DB 6	DDH		Port number for 8251A Number of bytes
0096	000000	00502	DB O	,0,0	;	Dummy bytes to get chip ready
0099	42	00503		100\$0010B	;	Reset and raise DTR
009 A	6E	00504 00505	DB 0	1\$10\$11\$10B		1 stop, no parity, 8 bits/char, divide down of 16
009B	25	00506	DB O	010\$0101B		RTS high, enable Tx/Rx
_		00507			. .	
009C	DE	00508 00509	DB 0	DFH		data for an 8253 chip Port number for 8253 mode
0090		00510	DB 1		;	Number of bytes to output
009E		00511		0\$11\$011\$0B	;	Select:
		00512 00513			;	Counter 2 Load LS byte first
		00513			;	Mode 3, binary count
009F		00515		DEH		Port number for counter
	02	00516	DB 2		:	Number of bytes to output

Figure 8-10. (Continued)

	00517	D1\$Baud\$Rate\$	Constant:		
00A1 3800	00518	DW	0038H		;1200 baud (based on 16x divider)
00A3 00	00519	DB	0		;Port number of 00 terminates stream
	00520				
	00521	D2\$Initialize		;Exa	Imple data for an 8251A chip
OOA4 DD	00522	DB	ODDH		;Port number for 8251A
00 A5 06	00523	DB	6		Number of bytes
00000 600000	00524	DB	0,0,0		;Dummy bytes to get chip ready
00A9 42	00525	DB	0100\$00		Reset and raise DTR
00AA 6E	00526	DB	01\$10\$1	1\$10B	;1 stop, no parity, 8 bits/char,
	00527				; divide down of 16
00AB 25	00528	DB	0010\$01	01B	;RTS high, enable Tx/Rx
	00529				
	00530			;Exe	ample data for an 8253 chip
OOAC DF	00531	DB	ODFH		;Port number for 8253 mode
00AD 01	00532	DB	1		;Number of bytes to output
00AE F6	00533	DB	11\$11\$0	11\$0B	;Select:
	00534				; Counter 3
	00535				; Load LS byte first
0015 55	00536				; Mode 3, binary count
OOAF DE	00537	DB	ODEH		;Port number for counter
00B0 02	00538	DB	2		;Number of bytes to output
0001 0000	00539	D2\$Baud\$Rate\$			
00B1 3800	00540	DW	0038H		;1200 baud (based on 16x divider)
00B3 00	00541	DB	0		;Port number of 00 terminates stream
	00542				
	00543				· · ·
	00544 00545	; This	for each	caule 15 USEC	l to determine the maximum sition in the ASCII time
	00546				. Note this table is
	00547				on block so that the clock
	00548	; can be	, iong tern s set "pern	manentlý" to	either 12 or 24 hour format.
	00549	; can be	. set per		erther is on an noun format.
	00550		The table	is processed	l backwards to correspond
	00551		the ASCII		, succurates to correspond
	00552				e value for the corresponding
	00553	; charad	ter in th	e ASCII time	at which a carry-and-reset-to-zero
	00554	; should	l occur.		
	00555	;			
00B4 00	00556	DB	0	;"Te	rminator"
	00557	CB\$12\$24\$Clock			
00B5 3334	00558	DB	^34	;Cha	inge to 1231 for a 12-hour clock
00B7 FF	00559	DB	OFFH	;"SI	ip" character
00B8 363A	00560	DB	'6:'		imum minutes are 59
OOBA FF	00561	DB	OFFH		ip" character
00BB 363A	00562	DB	·6: ·		imum seconds are 59
	00563	Update\$Time\$Er	nd:	;Use	d when updating the time
	00564	;			
	00565	; 			- New All and the balance
	00566		pies for t	ne real time	clock and watchdog
	00567	; timer			
	OOE/C				
AARD OC	00568	;	· · · · · · · · ·	DD /2	
OOBD 3C	00569	; RTC\$Ticks\$per	Second	DB 60	Number of real time clock
	00569 00570	; RTC\$Ticks\$per\$; ticks per elapsed second
OOBD 3C Oobe 3C	00569 00570 00571	;		DB 60 DB 60	; ticks per elapsed second ;Residual count before next
OOBE 3C	00569 00570 00571 00572	; RTC\$Ticks\$per\$ RTC\$Tick\$Count	:	DB 60	; ticks per elapsed second ;Residual count before next ; second will elapse
	00569 00570 00571 00572 00573	; RTC\$Ticks\$per\$:		; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574	; RTC\$Ticks\$per\$ RTC\$Tick\$Coun\$ RTC\$Watchdog\$(Count	DB 60 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(0 = no watchdog timer set)
OOBE 3C	00569 00570 00571 00572 00573 00574 00575	; RTC\$Ticks\$per\$ RTC\$Tick\$Count	Count	DB 60	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(O = no watchdog timer set) ;Address to which control
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574 00575 00576	; RTC\$Ticks\$per\$ RTC\$Tick\$Coun\$ RTC\$Watchdog\$(Count	DB 60 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(O = no watchdog timer set) ;Address to which control ; will be transferred if the
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574 00575 00576 00577	; RTC\$Ticks\$per\$ RTC\$Tick\$Coun\$ RTC\$Watchdog\$(Count	DB 60 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(O = no watchdog timer set) ;Address to which control
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574 00575 00576 00577 00578	; RTC\$Tick\$\$per4 RTC\$Tick\$Coun4 RTC\$Watchdog\$(RTC\$Watchdog\$/	Count	DB 60 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(O = no watchdog timer set) ;Address to which control ; will be transferred if the
ООВЕ ЗС ООВЕ ОООФ	00569 00570 00571 00572 00573 00574 00575 00576 00577 00578 00579	; RTC\$Ticks\$per4 RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$/ ;	Count Address	DB 60 DW 0 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(O = no watchdog timer set) ;Address to which control ; will be transferred if the
ООВЕ ЗС ООВЕ ОООФ	00569 00570 00571 00572 00573 00574 00575 00576 00577 00578 00579 00580	; RTC\$Tick\$\$per4 RTC\$Tick\$Coun4 RTC\$Watchdog\$(RTC\$Watchdog\$(; ; Functi	Count	DB 60 DW 0 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(O = no watchdog timer set) ;Address to which control ; will be transferred if the
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574 00575 00576 00577 00578 00579 00580 00580	; RTC\$Ticks\$per4 RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(; ; Functi	Count Address on key tab	DB 60 DW 0 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(0 = no watchdog timer set) ;Address to which control ; will be transferred if the ; watchdog count hits O
ООВЕ ЗС ООВЕ ОООФ	00569 00570 00572 00573 00574 00575 00576 00577 00578 00579 00580 00580	; RTC\$Tick\$\$per4 RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(; ; Functi ; This t	count address on key tab able cons:	DB 60 DW 0 DW 0 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(O = no watchdog timer set) ;Address to which control ; will be transferred if the
ООВЕ ЗС ООВЕ ОООФ	00569 00570 00571 00572 00573 00574 00575 00576 00577 00578 00579 00580 00580	; RTC\$Tick\$\$per4 RTC\$Tick\$Coun4 RTC\$Watchdog\$(RTC\$Watchdog\$(; ; Functi ; Functi ; This t ; folloo	Count Address on key tab	DB 60 DW 0 DW 0 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(0 = no watchdog timer set) ;Address to which control ; will be transferred if the ; watchdog count hits O
ООВЕ ЗС ООВЕ ОООФ	00569 00570 00571 00572 00573 00575 00576 00577 00578 00579 00579 00580 00581 00583	; RTC\$Tick\$Per4 RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(; ; ; Functi ; ; ; This t ; follow	count address on key tab able cons:	DB 60 DW 0 DW 0 DW 0	; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(0 = no watchdog timer set) ;Address to which control ; will be transferred if the ; watchdog count hits O
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574 00575 00575 00577 00578 00579 00580 00581 00582 00583	; RTC\$Tick\$\$per4 RTC\$Tick\$Coun4 RTC\$Watchdog\$(RTC\$Watchdog\$(; ; Functi ; Functi ; This t ; folloo	Count Address on key tak able cons ving struct	DB 60 DW 0 DW 0 DW 0	 ticks per elapsed second Residual count before next second will elapse Watchdog timer tick count (0 = no watchdog timer set) Address to which control will be transferred if the watchdog count hits O
ООВЕ ЗС ООВЕ ОООФ	00569 00570 00571 00572 00573 00574 00575 00576 00577 00578 00579 00580 00581 00581 00582 00583 00584	; RTC\$Tick\$\$per4 RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(; ; Functi ; This t ; follow ;	Count Address on key tak able cons ving struct	DB 60 DW 0 DW 0 DW 0 ole ists of a ser cure: Second chara terminal's f	 ticks per elapsed second Residual count before next second will elapse Watchdog timer tick count (0 = no watchdog timer set) Address to which control will be transferred if the watchdog count hits O
ООВЕ ЗС ООВЕ ОООФ	00569 00570 00571 00572 00573 00574 00575 00576 00577 00578 00580 00581 00581 00582 00583 00584	; RTC\$Tick\$Pers RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Count Address on key tak able cons ing struct DB	DB 60 DW 0 DW 0 DW 0 ole ists of a ser ture: Second chara terminal's f Third charac	 ticks per elapsed second Residual count before next second will elapse Watchdog timer tick count (0 = no watchdog timer set) Address to which control will be transferred if the will be transferred if the watchdog count hits O
ООВЕ ЗС ООВЕ ОООФ	00569 00570 00571 00572 00573 00574 00576 00576 00576 00577 00580 00581 00582 00583 00583 00584 00585 00584	; RTC\$Tick\$Pers RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(RTC\$Watchdog\$/ ; ; Functi ; ; This t ; follow ; ; ; (Count Address on key tak able cons ing struct DB	DB 60 DW 0 DW 0 DW 0 ole ists of a ser ure: Second chara terminal's f Third charac field will r	 ticks per elapsed second Residual count before next second will elapse Watchdog timer tick count (0 = no watchdog timer set) Address to which control will be transferred if the watchdog count hits O ies of entries, each one having the cter of sequence emitted by unction key ter of sequence NOTE: this
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574 00575 00576 00577 00578 00579 00580 00581 00582 00583 00583 00583 00584 00585 00588 00587 00588	; RTC\$Tick\$\$per4 RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(; Functi ; Functi ; This t ; follow ; ; ; (Count Address on key tak able cons ing struct DB	DB 60 DW 0 DW 0 DW 0 DW 0 ole ists of a ser ture: Second charac terminal's f Third charac field will r has been cor in function	<pre>; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(0 = no watchdog timer set) ;Address to which control ; will be transferred if the ; watchdog count hits O ies of entries, each one having the cter of sequence emitted by unction key ter of sequence NOTE: this) ot be present if the source code) figured to accept only two characters) key sequences.)</pre>
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574 00575 00576 00577 00578 00579 00580 00581 00582 00583 00584 00584 00585 00584 00585 00584	; RTC\$Tick\$Pers RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	Count Address on key tak able cons ing struct DB	DB 60 DW 0 DW 0 DW 0 DW 0 ole ists of a ser ture: Second charac terminal's f Third charac field will r has been cor in function	 ticks per elapsed second Residual count before next second will elapse Watchdog timer tick count (0 = no watchdog timer set) Address to which control will be transferred if the watchdog count hits O ies of entries, each one having the cter of sequence emitted by unction key ter of sequence NOTE: this) ot be present if the source code) figured to accept only two characters)
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574 00575 00576 00577 00580 00582 00582 00583 00582 00584 00585 00584 00585 00584 00585 00587 00588 00587 00588 00589 00590	; RTC\$Tick\$\$per4 RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(; ; Functi ; This t ; follow ; ; ; (; ; (; (; (Count Address on key tak able cons ing struct DB	DB 60 DW 0 DW 0 DW 0 DW 0 DW 0 DW 0 DW 0 DW	<pre>; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(0 = no watchdog timer set) ;Address to which control ; will be transferred if the ; watchdog count hits 0 ies of entries, each one having the cter of sequence emitted by unction key ter of sequence NOTE: this) ot be present if the source code) figured to accept only two characters) key sequences.) the equates for:) }</pre>
ООВЕ ЗС ООВГ ОООф	00569 00570 00571 00572 00573 00574 00575 00576 00576 00578 00578 00580 00581 00582 00583 00584 00583 00584 00585 00585 00585 00587	; RTC\$Tick\$Pers RTC\$Tick\$Count RTC\$Watchdog\$(RTC\$Watchdog\$(; Functi ; Functi ; This t ; follow ; ; ; (; (; (; (; (; (; (Count Address on key tak able cons ing struct DB	DB 60 DW 0 DW 0 DW 0 DW 0 DW 0 DW 0 DW 0 DW	<pre>; ticks per elapsed second ;Residual count before next ; second will elapse ;Watchdog timer tick count ;(0 = no watchdog timer set) ;Address to which control ; will be transferred if the ; watchdog count hits 0 ies of entries, each one having the cter of sequence emitted by unction key ter of sequence NOTE: this) of be present if the source code) figured to accept only two characters) key sequences.) the equates for:)</pre>

	00	0594	:				
			•	DD	A		vine to be formed into the concole
		0595	;	DB			ing to be forced into the console
		0596	;				en the corresponding function key
	00)597	;		is pres	sed. The	e last byte of this string must be
1	00	598	;		00H to	terminat	e the forced input.
)599	•				
001B		0600	, Function\$Key\$L	ead 💊	EQU	1BH	;Signals function key sequence
0003		0601	Function\$Key\$L		EQU	3	;Number of characters in function
0003			Function #Key#L	ength	EGO	3	
1		602					; key input sequence (NOTE: this
1	00	603					; can only be 3 or 2 characters).
	00	0604					
	00	0605	:				
		606	•				The logic associated with function:
1		0607					; key recognition is made easier with
1							
		0608	T A O A			500	; the following equate
0001		0609	Three\$Characte	rprunctic	יח	EQU	Function\$Key\$Length - 2
		0610					Character\$Function will be TRUE if the
(00	0611				; func	tion keys emit a three character
	00	0612				; sequ	ence, FALSE if they emit a two character
		0613					ience.
		0614					
I					h		the erms length of defined by:
		0615	; Each e	nery in t	ne table	must be	the same length, as defined by:
		0616	;				· · · · · · · · · · · · · · · · · · ·
0013	= 00	0617	CB\$Function\$Ke	y\$Entry\$S	ize	EQU	16 + 1 + Function\$Key\$Length - 1
	00	0618	;				* * *
1		0619	;				
		0620	;	Maximum	length	of subst	itute Lead character is not
		0621	÷	string		Jubst	in table entry
1				second			For the terminating 00H
1		0622					to the terminating you
		0623	;				
1		0624	; The la	st entry	in the t	able is	marked by a 00-byte.
1		0625	;				
1	00	0626	; The ex	ample val	ues show	vn below	are for a VT-100 terminal.
1		0627	;				
1		0628	CB\$Function\$Ke	v\$Table:			
1		0629	•		123454	789 1234	↓ 5 67 <- Use to check length
0000			, DD	101 101			
	4F5046756E00		DB	- 101, 1P1	, Functi	ton Key 1	L', LF, 0, 0
	4F5146756E00		DB	U, Q	, Functi	ion Key 2	2′,LF,0,0
	4F5246756E00		DB	101,1R1	, 'Functi	ion Key 3	3', LF, 0, 0
OOFC	4F5346756E00	0633	DB	/0/,/S/	', 'Functi	ion Key 4	¥',LF,0,0
1	00	0634	;				
1		0635	•		123456	5789.1	
DIOF	584155702000		, DB	·[· . ·] ·),0,0,0,0,0,0,0
0122	5B42446F7700	0637	DB		Down 4	Arrow 15	F, 0, 0, 0, 0, 0, 0
			DB		/ Diate	Arrent 1	_F,0,0,0,0,0,0
	5B4352696700						
0148	5B444C656600		DB	ι, D'	, Lett A	arrow', LH	F, 0, 0, 0, 0, 0, 0 `
1		0640					
	000000000000000000000000000000000000000		DB),0,0,0,0,0,0,0 ;Spare entries
016E	000000000000000000000000000000000000000	0642	DB	0,0,0,0	0,0,0,0,0	0,0,0,0,0	0,0,0,0,0,0,0,0
	000000000000000000000000000000000000000		DB				0, 0, 0, 0, 0, 0, 0
	000000000000000000000000000000000000000		DB				0,0,0,0,0,0,0,0
			DB), 0, 0, 0, 0, 0, 0, 0
	000000000000000000000000000000000000000						
	000000000000000000000000000000000000000		DB				0, 0, 0, 0, 0, 0, 0
	000000000000000000000000000000000000000		DB				0, 0, 0, 0, 0, 0, 0, 0
01E0	000000000000000000000000000000000000000	0648	DB),0,0,0,0,0,0,0
01F3	000000000000000000000000000000000000000	0649	DB	0,0,0,0	,0,0,0,0	0,0,0,0,0	0,0,0,0,0,0,0,0
	000000000000000000000000000000000000000		DB	0.0.0.0	0.0.0.0.0	5,0,0,0,0	0,0,0,0,0,0,0,0
1 0200		0651					
0219		0652	DB	OFFH.OF	FH	Termir	nator for utility that preprograms
1 0217			00	or chi, Or		,	ction key sequence
1		0653				, iuno	LICH KEY BEAMENCE
1		0654	;				
1		0655	;				
	00	0656	; Consol	e output	escape s	sequence	control table
	00	0657	;				
1		0658	; Thist	able is m	reference	ed after	a Function\$Key\$Lead character
1		0659	; has be	en detect	ted in th	ne CONQUI	T routine. The next character
1		0660					compared to the first byte
1		0660	; to be ; in eac		table of	1991 19 1991 19	a match is found, then control
1			, in eac	n o uyte	to the	nci7. Al	following the byte that matched.
1		0662	; is tra	usierred	to the a	auuress 1	iorrowing the byte that matched.
1		0663	;				
1		0664	CONOUT\$Escape\$				
021B	74 00	0665	DB	1t1		;Read o	current time
0210		0666	DW	CONOUT	₿Time		
021E		0667	DB	'd'		:Read /	current date
021E			DB DB	CONOUTS	tint o	,nead (Larrent wate
		0668	~		puate		
0221	/5 00	0669	DB	~u~			⊔rrent time
0222	5004 00	0670	DW	CONDUTS	≸Set\$Time	e	

0224 65	00671	DB	'e'	:Set	current date
0225 4E04	00672	DW	CONOUTS	≸Set\$Date	
	00673				
0227 00	00674	DB	0	• Term	linator
0227 00	00675		· ·	,	(index)
	00676	, Long\$Term\$CB\$Er	d.		
1	00677	;			
	00800	;#			
	00801	;			
	00802	; Interru	ipt vecto	or	
	00803	;			
	00804				y the programmable interrupt
	00805	; control	ler a	an Intel 8259A.	
	00806	;			
	00807	NOTE: T	he inter	rupt controlle	r chip requires that the
	00808	;	interr	pt vector tabl	e start on a paragraph
1	00809	;			ieved by the following ORG line
0240	00810	ORG		OFFEOH) + 20H	
0210	00811	Interrupt\$Vecto		off control - zon	
	00812	Interruptivecto	•••	;Interrupt nu	mbor
0240 C37808	00813	JMP	DTC#T-		;0 clock
				terrupt	
0243 00	00814	DB	0		;Skip a byte
0244 C3E806	00815	JMP		ter\$Interrupt	;i character I/O
0247 00	00816	DB	0		- · ·
0248 C3D80E	00817	JMP		Interrupt	;2 not used
024B 00	00818	DB	0		
024C C3D80E	00819	JMP	Ghost\$1	Interrupt	;3 not used
024F 00	00820	DB	0		
0250 C3D80E	00821	JMP	Ghosts	Interrupt	;4 not used
0253 00	00822	DB	0		
0254 C3D80E	00823	JMP		Interrupt	;5 not used
0257 00	00824	DB	0		, o net use
0258 C3D80E	00825	JMP	-	Interrupt	;6 not used
025B 00	00826	DB	0	ancerrapt	,o not dsed
			-		
025C C3D80E	00827	JMP	GNOSTPI	Interrupt	;7 not used
	00828	;			
	00900	;#			
	00901				
	00902	; Device	port num	mbers and other	equates
	00903	;			
0080 =	00904	CIO\$Base\$Port	EQU	80H	;Base port number
	00905				
0080 =	00906	DO\$Base\$Port	EQU	CIO\$Base\$Port	;Device O
0080 =	00907	D0\$Data\$Port	EQU	DO\$Base\$Port	
0081 =	00908	D0\$Status\$Port	EQU	D0\$Base\$Port	+ 1
0082 =	00909	D0\$Mode\$Port	EQU	D0\$Base\$Port	
0083 =	00910	D0\$Command\$Port		DO\$Base\$Port	
0000 -	00911	;	Luo		
	00912				
0084 =	00913	D1\$Base\$Port	EQU	CIO\$Base\$Port	+ 4 ;Device 1
0084 =	00913	DI\$Data\$Port	EQU	D1\$Base\$Port	· - Instruct
0084 =	00914				
		D1\$Status\$Port	EQU	D1\$Base\$Port	
0086 =	00916	D1\$Mode\$Port	EQU	D1\$Base\$Port	
0087 =	00917	D1\$Command\$Port	EQU	D1\$Base\$Port	+ 3
1	00918				
0088 =	00919	D2\$Base\$Port	EQU	CIO\$Base\$Port	+ 8 ;Device 2
0088 =	00920	D2\$Data\$Port	EQU	D2\$Base\$Port	
0089 =	00921	D2\$Status\$Port	EQU	D2\$Base\$Port	
008A =	00922	D2\$Mode\$Port	EQU	D2\$Base\$Port	+ 2
008B =	00923	D2\$Command\$Port	EQU	D2\$Base\$Port	
	00924				
004E =	00925	D\$Mode\$Value\$1	EQU	01\$00\$11\$10B	
1	00926				op bit, no parity
1	00927				ts, Async. 16x rate
003C =	00927	D\$Mode\$Value\$2	FOU	,8 D1 00\$11\$1100B	ta, mayne, iox rate
0000 -	00928	Denoue avaideas	200		w on internal alcak
1					x on internal clock
00037 -	00930	DeCourse dell'	FOU		baud
0027 =	00931	D\$Command\$Value	EQU	00\$100111B	
1	00932				al mode
	00933				le Tx/Rx
1	00934				and DTR active
0038 =	00935	D\$Error	EQU	0011\$1000B	
0037 =	00936	D\$Error\$Reset	EQU	00\$110111B	
1	00937				as command value plus error reset
0001 =	00938	D\$Output\$Ready	EQU	0000\$0001B	
0002 =	00939	D\$Input\$Ready	EQU	0000\$0010B	
0080 =	00940	D\$DTR\$High	EQU	1000\$0000B	;Note: this is actually the

Figure 8-10. (Continued)

0007 : dist-set-ready pin 0007 00074 DefairestTS EQU 0001001118 FairestTS. Tr/Re enable 0007 00074 DefairestTS EQU 00010001118 FairestTS. Tr/Re enable 0007 00074 DefairestTS EQU 0010001118 FairestTS. Tr/Re enable 0007 00074 Interrupt controller ports (Intel 525%) 00074 Interrupt controller ports (Intel 525%) 0007 00076 Note : these stats are placed here so that they 00076 0007 000761 Interrupt controller ports (Intel 525%) 0008 000781 Interrupt controller ports (Interrupt contrupt (Interupt controller ports (Interrupt controller p							
00943 i. to the DIR pin on the cable 0007 = 00944 DBDroskTS EQU 0001400111B :Dros NT, Tr.Km. enable 0007 = 00945 DBDroskTS EQU 0001400111B :Dros NT, Tr.Km. enable 0008 = 000945 : Interrupt controller ports (Intel 525%A) 0008 = 000951 : Note : these secures are placed here so that they 00095 : : ind thus avoid f" (Dhase) errors in ASM. 0008 = 000953 : : 0008 = 000953 : : 0008 = 000953 : : : 0008 = 000953 : : : 0008 = 000953 : : : 0008 = 000953 : : : 0008 = 00953 : : : 00054 = 000551 : : : 00055 = : : : : : 00056 : : :				-			
0027 = 00944 DPRaiseRTS EQU 000:400111B ;RaiseRTS, Tx/Rx enable 0097 = 00945 DPCoeRTS EQU 000:400111B ;RaiseRTS, Tx/Rx enable 00984 ; 00984 ; 00985 ; 00984 ; 00985 ; 00986 ; 00986 ; 00986 ; 00986 ; 00986 ; 00986 ; 00986 ; 00986 ; 00986 ; 00987 ; 00986 ; 00986 ; 00986 ; 00987 ; 00986 ; 00986 ; 00986 ; 00986 ; 00986 ; 00987 ; 00986 ; 00987 ; 00986 ; 0							
0007 = 00945 DBDrosBRTS EGU 00900001118 ;Drop RTS, Tx/Rx enable 000745 ; Interrupt controller ports (Intel 8259A) 000750 ; Note : these equates are placed here so that they 000751 follow the definition of the interrupt vector 000752 ; and thus avoid P ⁻ (phase) errors in AB. 000753 ; Context and thus avoid P ⁻ (phase) errors in AB. 000753 ; Context and thus avoid P ⁻ (phase) errors in AB. 000751 ; Context and thus avoid P ⁻ (phase) errors in AB. 000752 ; and thus avoid P ⁻ (phase) errors in AB. 000753 ; Context and result control word 1 000753 ; Context and result control word 2 000753 ; Context and result control word 2 000754 ; EGU ODFH (pherational control word 2 000754 ; EGU ODFH (pherational control word 2 000755 ; Context and result context a	0007 -			FOU	004140		
00046 00047 00047 00050	0007 =						Drop RTS. Tx/Rx enable
00047 interrupt controller ports (intel \$259A) 00049 Note : these equates are placed here so that they 00051 Note : these equates are placed here so that they 00052 and thus avoid 'P' (phase) errors in ASN. 00053 and thus avoid 'P' (phase) errors in ASN. 00058 000553 00058 00057 00058 00057 00059 00057 00050 00057 00050 00057 00050 00057 00050 00057 00052 00058 00050 00057 00051 00584 00052 00052 00052 1CSICWIFFORT 00052 1CSICWI 00053 1CSICWI 00054 1CSICWI 00055 1CSICWI 00056					004040		
000949 i Note : these equates are placed here so that they 00095 i follow the definition of the interrupt vector 0009 000954 ICS0CM14Fort EQU 00964 ics0cm14Fort 0008 000955 ICS0CM14Fort EQU 00964 ics0cm14Fort EQU 00964 ics0cm14Fort 0008 000955 ICS0CM14Fort EQU 00964 initialization control word 1 0009 000956 ICS1CHU14Fort EQU 00964 initialization control word 2 0020 000950 ICS1CHU14Fort EQU 00964 initialization control word 2 0020 000950 ICS1CHU1 EQU 2004 ihomsecitic end of interrupt 0020 000956 Ics1CHU1 EQU 2004 ihomsecitic end of interrupt 0056 ICS1CHU EQU 2004 ihomsecitic end of interrupt 0056 ICS1CHU EQU 2004 ihomsecitic end of interrupt 0056 ICS1CHU EQU 1hterrupt Vector AND ihomsecitic end interrupt 0057 ICS1CHU EQU ihterrupt Vector AND </td <td></td> <td>00947</td> <td>;</td> <td></td> <td></td> <td></td> <td></td>		00947	;				
00050 j. Note i these equates are placed here so that they 00051 follow the definition of the interrupt vector 00052 and thus avoid ?? (phase) errors in ASM. 00058 000551 00058 000571 00058 000571 00059 000571 00059 000571 00059 000571 00050 000571 00050 000571 00050 000591 00050 000501 00050 000501 00050 000501 00050 000501 00050 000501 00050 000501 00050 000501 00051 00051 00052 000501 00053 00051 00054 000521 00055 00051 00052 000521 00052 000521 00052 000521 00052 000521 00052 000521 00052 000521 00052 000521 </td <td></td> <td></td> <td>; Interru</td> <td>upt comtr</td> <td>oller p</td> <td>orts (In</td> <td>tel 8259A)</td>			; Interru	upt comtr	oller p	orts (In	tel 8259A)
00951 ; follow the definition of the interrupt vector 00953 ; 0008 00953 ; interrupt vector 0008 00955 ; itSOCUISPORT EDU 008H ;Oberational control word 1 0008 00955 ; itSOCUESPORT EDU 008H ;Oberational control word 2 0009 00956 ; itSOCUESPORT EDU 008H ;Oberational control word 1 0009 00958 ; itSOCUESPORT EDU 008H ;Initialization control word 1 0000 00958 ; itSEtS the A7 - A5 bits of the interrupt 0056 = 00962 ; itSEtS the A7 - A5 bits of the interrupt 0056 = 00962 ; itSetS the A7 - A5 bits of the interrupt 00664 ; ; itSetS the A7 - A5 bits of the interrupt 00664 ; ; itSetS the A7 - A5 bits of the interrupt 00666 ; ; ; itSetS the A7 - A5 bits of the interrupt 00670 ; ; ; itsetSetS the S15 - A6 of the interrupt 00671 ; ; itsetruptVector SHR 3 itsets the first structure in 00672 ; ; ; ithe long term configuration block 00773 ; ; ; ithe long term configuration block 00775 ; ita0101 ;			;				
00052 and thus avoid 'P' (shase) errors in ASH. 00059 00054 IC40CH18Port EGU 00084 ;0perational control word 1 0005 00057 IC40CH18Port EGU 00084 ;0perational control word 2 0005 00057 IC41CHL2Port EGU 0084 ;1nitialization control word 2 0005 00057 IC41CH18Port EGU 00844 ;1nitialization control word 2 0005 00057 IC41CH18Port EGU 00844 ;1nitialization control word 2 00050 00059 ; EGU 1014 intraution control word 2 00056 00054 ; vector address pluss 00056 ; vector address plus 00056 ; vector is Harsed 000571 ; vector is Harsed 000772 ; vector is Harsed 000773 ; interrupt 0 (clock) enabled 000774 ; interrupt 0 (clock) enabled 000775 ; interrupt 0 (clock) enabled 000776 ; vutual formal contral word i interrupt 000776 ; interrupt 1 (character input) enabled 000777 ; interrupt 1 (character input) enabled 000778 ; interrupt 1 (character input) enabled 000777 ; interrupt 1 (clock) enabled 0007							
00093 ; 00094 ;0Perational control word 1 0008 00095 [100C014Port EDU 0084] ;0Perational control word 2 0008 00095 [100C014Port EDU 0084] ;0Perational control word 2 0009 00095 [100C014Port EDU 0084] ;initialization control word 2 0020 00960 [100C014Port EDU 0084] ;initialization control word 2 0020 00960 [100C016] EOU 0084] ;initialization control word 2 0020 00960 [100C016] EOU 0084] ;initialization control word 2 0056 00962 [100C016] EOU 0084] ;initialization control word 2 0056 00962 [100C016] ;interrupt Wetcor AMD 1110600008] ;000810108 00964 : ************************************			,				
0009 = 00054 [C00C014Port EOU 009H ; 0perational control word 1 0008 = 00055 [C00C043Port EOU 008H ; 0perational control word 3 0008 = 00055 [C00C043Port EOU 008H ; 10tialization control word 1 0009 = 00059 [C01C043Port EOU 008H ; 10tialization control word 1 0020 = 00050 [C01C043Port EOU 008H ; 10tialization control word 1 0020 = 00050 [C01C041] EOU 008H ; 10tialization control word 1 0055 = 00050 [C01C041] EOU 008H ; 10tialization control word 1 0056 = 00050 [C01C041] EOU 008H ; 10terrupt%vector AND 110060008H + 000610110B ; 5%ts the A7 - A5 bits of the interrupt 0056 = 00056 ; 10terrupt%vector SHR 8 00056 00056 = 00057 ; 10terrupt%vector SHR 8 00077 00057 = ; vector address, Note the interrupt 00077 00077 = ; vector address, Note the interrupt 00077 00077 = ; interrupt%vector SHR 8 00077 00077 = ; vector address, Note the interrupt 00077 00077 = ; interrupt%vector SHR 8 000777			,	and tha	s avoiu	r (Pi	ase/ errors in Hon.
0008 = 00955 IC40CL22Port EBU 0084 ; 0Derational control word 2 0008 = 00957 IC4CL22Port EBU 0084 ; 1nitialization control word 3 0009 = 00950 IC4CL22Port EBU 0084 ; 1nitialization control word 2 0020 = 00950 IC4CL21Port EBU 0084 ; 1nitialization control word 2 0056 = 00952 IC4CL2 EDU 204 ; Nonspecific end of interrupt 0056 = 00952 IC4ICL2 EDU 204 ; Nonspecific end of interrupt 0056 = 00952 IC4ICL2 EDU (InterruptWector AND lilo60008) + 00051010B ; 5555 ; AAT - AB of the interrupt 0056 = 00957 IC4ICL2 EDU InterruptWector SHR 3 00957 ; No IC4A needed 00572 = 00957 ; IC40CH1 EDU 111191100B ; interrupt mask : interrupt i mask 00574 ; No IC4A needed 00976 ; No VA AM : interrupt i ask : interrupt i ask 00575 ; IC40CH1 EDU 111191100B ; interrupt mask : interrupt i ask 00576 ; NOV A.M : interrupt 0 (clock) enabled : interrupt i ask	00D9 =		IC\$OCW1\$Port	EQU	0D9H	;Opera	tional control word 1
0008 = 00957 ICSICULISFORT EQU 00984 ;Initialization control word 2 0020 = 00950 ; Nonspecific end of interrupt 0056 = 00962 ICSICH EQU 20H ;Nonspecific end of interrupt 0056 = 00963 ICSICH EQU 20H ;Nonspecific end of interrupt 0056 = 00964 ;Sist the A7 - A5 bits of the interrupt ; vector address plus: 00966 : ;Sist the A7 - A5 bits of the interrupt ; vector address plus: ; 00967 : ;Single 2259 in system ; No ICMA meeded 00067 : ;vector address plus: ;vector address plus: ;vector address plus: 00071 : ;vector is the first structure in ;vector address plus: ;vector address plus: 00077 : ;vector address plus: ;interrupt slucture in ;vector address plus: 00978 : interrupt slucture slucture in ;vector address plus: ;vector address plus: 00976 : ;interrupt l (clack) enailed ;interrupt l (clack) enailed 00976 : ;interrupt l (clack) enailed ;vector address plus: ;vector address plus: 010101 : <td< td=""><td>00D8 =</td><td>00955</td><td>IC\$OCW2\$Port</td><td>EQU</td><td>ODSH</td><td></td><td></td></td<>	00D8 =	00955	IC\$OCW2\$Port	EQU	ODSH		
0009 = 00958 00950 ; 00950 ; 00950 ; 00950 ; 00950 ; 00951 ; 00955 ; 00955 ; 00955 ; 00955 ; 00955 ; 00955 ; 00956 ; 00957 ; 00958 ; 00957 ; 00958 ; 00957 ; 00958 ; 00057 ; 00958 ; 00050 ; 00057 ; 0005 ; 00005 ; 0005 ; 0005 ; 00005 ; 0005 ; 0005 ; 0005 ; 0005							
0020 = 00950 00961 00964 00964 00964 00964 00965 ICEEDI EQU EQU (InterruptSVector AND ILIOS0000) + 000010108 15 stothe A7 - A5 bits of the interrupt 15 vector address plus of the interrupt 15 vector address plus atternant 00966 00966 0002 = 00967 00967 00967 IEdge triggered 15 single 8259 in system 00968 0002 = 00969 00970 00977 InterruptVector SNR 3 - A6 of the interrupt 1 vector is the first structure in 17 vector is the first structure in 18 vector is the first structure in 19 vector is the interrupt i (character input) enabled 19 vector is the structure in 19 vector is the interrupt 10 vector is the vector is in the 10 vector is the vector is in the 10 vector is 10 vector is in the 10 vector is 10							
0020 = 00%00 IC&EDI EBU 20H ; Nonspecific end of interrunt 0056 = 00%21 ICICW1 EDU (Interrupt@vector AND 1110%00008) + 000%10108 00%64 ; vector address plus ; vector address plus ; vector address plus 00%64 ; vector address plus ; vector address plus ; vector address plus 00%67 ; single 8259 in system ; No ICM4 needed 000%7 ; vector skff 8 ; vector address. Note the interrunt 00%76 ; vector interrunt fiveration block ; vector interrunt interrunt 00%77 ; vector interrunt mask ; vector interrunt interrunt 00%76 ; interrunt 0 (clock) enabled ; interrunt 1 (character input) enabled 00%77 ; interrunt 1 (character input) enabled ; interrunt interrunt input) enabled 00%77 ; interrunt 1 (character input) enabled ; interrunt input) enabled 00%77 ; interrunt interrunt input) enabled ; interrunt input) enabled 00%76 ; interrunt input) enabled ; input) enabled 00%77 ; interrunt input) enabled ; input) enabled 00%77	0009 =		IC\$ICW2\$Port	EQU	ODAH	;initi	alization control word 2
0056 =00961 00963 00965 00966 00966ICSICHEGU(Interrupt@vector AND 1110@0000B) + 000%10110B iSets the A7 - A5 bits of the interrupt i vector address plus; i Edge triggered 009660002 =00968 00968 00968 00968 009690 00971 00972 00972 00972 00972 00977 00973 00977 00	0020 =		; IC\$EDI	FOL	20H	• Nonso	ecific end of interrupt
0056 = 00962 IC\$ICHI EQU (Interrupt\$Vector AND 110000008) + 0000101108 00964 ; vector address plus: 00966 ; vector address plus: 00966 ; address plus: 00966 00967 ; bit bit AF - AS bits of the interrupt 00968 ; bit bit AF - AS of the interrupt 00971 ; vector address. Note the interrupt 00973 ; vector is the first structure in 00974 ; vector is the first structure in 00975 ; vector is the first structure in 00977 ; interrupt (chock) enabled 00978 ; ; 0100 ; interrupt (chock) enabled 01010 ; interrupt (chock) enabled 0102 ; ; 0103 isplay\$Message: ; interrupt (chock) enabled 0255 7E 01105 ; ;	0020 -		;	200	2011	, NO.11 2F	
00064; vector address plus: 0006600066; 4-byte interval 0006700067; 5ingle 8259 in system i No IC44 needed00068IC\$ICW200069IC\$ICW200069IC\$ICW200070; No IC44 needed00071; Address bits Al300072; Address bits Al300073; Intervut Wector SHR 8 ; No IC44 needed00074; Theoret SHR 9 ; No IC44 needed00775; Intervut 1 ; Intervut 100776; Intervut 1 ; Intervut 100777; Intervut 1 ; Intervut 100778;00777; Intervut 1 ; Intervut 100777; Intervut 1 ; Intervut 100777; Intervut 1 ; Intervut 100777; Displays the spcified message on the console. ; intervut 1 ; Intervut 101103;01104; Sutput. A OOH-byte terminates the message. ; Det next message byte0255 7E01106MOV ; A, M0264 CB01107ORA ; Save message pointer0265 E0110PUSH ; H ; Fave ressage pointer0264 CB01111CALL ; CONDUT ; Save message pointer0264 CB01120iff ; Intervut 2 ; Det next message pointer0265 E0110PUSH ; This routine is entered either from the cold or warm ; Base page, and also sets the high-level disk driver's ; Det code. It sets up the JHA address].0264 CB01200; A ; Intervut 4dress it the MA address].0265 CB01207; A ; Dot code.	0056 =		IC\$ICW1	EQU	(Intern		
00965 ; Edge triggered 00966 ; A-byte interval 00967 Single 8259 in system 00968 i No ICM4 needed 00071 ; Vactor SHR 8 00072 ; vector address, Note interrupt 00073 ; vector is the first structure in 00074 ; vector is the first structure in 00075 IC\$OUT EOU Interrupt O (clock) enabled 00077 ; interrupt O (clock) enabled ; 00077 ; interrupt O (clock) enabled ; 01001 ; ; interrupt O (clock) enabled ; 01010 ; ; interrupt O (clock) enabled ; 01101 ; ; interrupt O (clock) enabled ; 0257 01105 ; interrupt O (clock) enabled ; 0260 interrupt O (clock) enabled ; ; ; 0257 0110 DEA ; ; ;							
00966 00968;4-byte interval is Single 8259 in system No IC44 needed0002 =00969 00970 00971 00972 00972 00973 00973 00973 00974 00973 00974 00974 00975EGUIntervietWeetor SHR 8 iAddress bits A15 - A8 of the intervipt i vector address. Note the intervipt i vector is the first structure in 00974 i the long term configuration block00FC =00975 00976 00976 00976 00977 001100 00977 001100 001100 001101 001101 001101 001102 001100 001101 001102 001100 001101 001100<						; vec	
00967 ; Single 8259 in system 0002 = 00969 IC\$ICW2 EGU InterruptWeetor SHR 8 00071 ; Vector address bits AIS - A6 of the interrupt 00973 ; vector is the first structure in 00974 ; vector is the first structure in 00977 ; interrupt 0 (lock) enabled 00977 ; interrupt 0 (lock) enabled 00977 ; interrupt 0 (lock) enabled 0100 ;# ; interrupt 0 structure in 01101; ; interrupt 0 telses interrupt 0 structure in 01102; ; ; interrupt 0 structure in 0256 7E 01107 MOV A for structure in 0257 7E 01107 MOV A for structure in 0256 7E 01107 MOV A ifferentiation 0252 24F 01107 MOV C.A iffer							
0092 = 00950 00970 00971 istructure istructure SHR 8 istructure in the interrupt istructure interrupt istructure interrupt istructure interrupt istructure interrupt istructure interrupt istructure interrupt istructure interrupt istructure istructure interrupt istructure istructure interrupt istructure istructure istructure interrupt istructure istructure istructure istructure istructure istructure istructure istructure istructure interrupt istructure istructure istructure interrupt istructure istructure istructure istructure interrupt istructure istructure istructur						;	
0002 = 00969 IC\$ICW2 EGU InterruptWeetor SHK 8 00071 : vector address bits A15 - A8 of the interrupt 00971 : vector address bits A15 - A8 of the interrupt 00973 : vector address bits A15 - A8 of the interrupt 00974 : vector is the first structure in 00975 : vector address bits A15 - A8 of the interrupt 00976 : vector address bits A15 - A8 of the interrupt 00977 : the long term configuration block 00976 : Interrupt 0 (clock) enabled 00977 : Interrupt 1 (character input) enabled 00977 : Displays the specified message on the console. 01100 :# 01101 : 01102 : 01103 Displays the specified message on the console. 01104 : 0255 T 01105 MOV A.M 0265 T 02105 : 0225 01105 MOV 0264 1111 CAL 0265 0110 PUSH 0266 01111 CAL 0						;	
00970; Address bits A15 - A8 of the interrupt00971; vector address. Note the interrupt00972; vector address. Note the interrupt00973; vector address. Note the interrupt00974; vector address. Note the interrupt00975IC\$0CWIEQU00976; Interrupt 0 (clock) enabled00977; Interrupt 0 (clock) enabled00978;01101; Interrupt 0 (clock) enabled01102;01103Display\$Message:01104; Dn entry. HL points to a.stream of bytes to be01105joutput. A 00H-byt terminates the message.0260D11070281C801108RZ02624111001109MOV0264C801109MOV0264C10108026501110026601111026701111026801112026901111026401112026901113026901113026901113026011131111Fist sets up the JMP instructions in the0269111502100; input/output address (the DM Address)022612000231; base page, and also sets the high-level dist hild024; bisplay\$Message02571111024; base page, and also sets the high-level dist hild025812000260120202601212 </td <td>0002 =</td> <td></td> <td>IC\$ICW2</td> <td>EQU</td> <td>Interro</td> <td></td> <td>or SHR 8</td>	0002 =		IC\$ICW2	EQU	Interro		or SHR 8
00972 ; vector is the first structure in 00974 ; the long term configuration block 00974 ; Interrupt 0 (clock) enabled 00976 ; Interrupt 0 (clock) enabled 00977 ; Interrupt 0 (clock) enabled 00978 ; 01101 ; 01102 ; 01103 Display\$Message: 01104 ; Display\$ the specified message on the console. 01105 ; Dutput, A 00H=byte terminates 0260 B7 01106 0261 C8 01108 0262 4F 01109 0263 E5 01110 0264 C EDD703 01111 CALL CONOUT 0265 762 01113 0264 C EDD703 01111 CALL CONOUT 0265 762 01114 0267 E1 01112 0268 E3 01101 0269 C335F02 01114 02112 Enter\$CPM: 1 ; This routine is entered either from the cold or warm 0265 320000 01205 1205 ; Interrupt if et BDA action 0005H		00970					
00973 ; the long term configuration block 00FC = 00975 ICSOCHI EOU 1111\$1100B ; Interrupt mask 00976 ; Interrupt 0 (clock) enabled ; Interrupt 1 (character input) enabled 00977 ; Interrupt 1 (character input) enabled 00978 ; 01100 ;# 01101 ; 01102 ; 01103 Display#Message: 01104 ;0n entry, HL points to a stream of bytes to be 01105 ;0n entry, HL points to a stream of bytes to be 0256 01106 MOV 0256 01107 ORA 0256 01108 RZ 0256 1110 PUSH 0254 EDD703 01111 0254 EDD703 01111 0254 EDD703 01114 0257 POP H ;Recover message pointer 0256 0257 01113 INX H 0269 C35F02 01114 JMP Display*Message ;Loop until comp						; vec	tor address. Note the interrupt
00FC = 00974 00976 CGOCH1 EOU 1111\$1100B :Interrupt mask :Interrupt 0 (clock) enabled 00976 00977 ;							
OOFC =OO975IIIsOCMIEOUIIIIsIIOOBfilterrupt 0 (clock) enabled (clock) enabled (clock) enabledOO976;;;OU977;;OU100;#;#OI101;;OI102;;OI103Display\$Message:;Displays the specified message on the console. ;On entry, HL points to a.stream of bytes to be ;Output. A OOH-byte terminates the message.O257 7EOI106MOVA.MO261 C8OI107ORAAO263 E5OI110PUSHHO264 C8OI108RZ;Yes, return to callerO264 C8OI109MOVC.A;Prepare for outputO264 C8OI111CALLCONOUTO265 C3OI110PUSHHSee over message pointerO264 C1OI112POPH;Recover message pointerO265 C35F02OI114JMPO115;O1200;#O1201;O1202Enter\$CPM:O1203; boot code. It sets up the jmh=vel disk driver's ; input/output address (the DMA address).O1204;joot code. It sets up the jmh=vel disk driver's ; input/output address (the DMA address).O274 210300O1205;O274 210300O1206;O274 210300O1213LXIO274 210300O1214LXIO277 220100O1215SHLDO277 220100O1215O217LXIB,80H<			-			; the	Tong term consiguration brock
00976:Interrupt 0 (clock) enabled00977;Interrupt 1 (character input) enabled00978;01100:#01101;01102i01103Display\$Message::Displays the spacified message on the console.01104:01105ioutput. A OOH-byte terminates the message.025F 7E0106MOV0260 B701107MOA0260 B701107MOA0261 C801108RZ0262 E4F01109MOV0264 CDD703011110264 CDD703011110264 CDD703011110264 CDD703011120267 F1011120110PUSH0268 23011130113INXHpDPH:Recover message pointer0269 C35F02011140113ix01202:#01203: iboot code. It sets up the JMP instructions in the01204: jboot code. It sets up the JMP instructions in the01205: input/output address (the DMA address).01206:01207MVI01206:01206:01207MVI01206:01206:01207MVI01208:01209:01206:01206:01207MVI01206:01206:01210:01206 </td <td>00FC =</td> <td></td> <td>IC\$OCW1</td> <td>EQU</td> <td>1111\$1</td> <td>100B</td> <td>;Interrupt mask</td>	00FC =		IC\$OCW1	EQU	1111\$1	100B	;Interrupt mask
<pre>00977 ; Interrupt 1 (character input) enabled 00978 ; 01100 ;# 01101 ; 01102 ; 01103 Display\$Message: ;Displays the specified message on the console. 01104 ;0n entry, HL points to a.stream of bytes to be 01105 ;00 entry, HL points to a.stream of bytes to be 01105 ;00 entry, HL points to a.stream of bytes to be 01105 ;00 entry, HL points to a.stream of bytes to be 01105 ;00 entry, HL points to a.stream of bytes to be 01106 MOV A, M ;0et next message byte 0260 B7 01107 ORA A ;Check if terminator 0263 E5 01110 PUSH H ;Steve message pointer 0264 CBD703 01111 CALL CONDUT ;60 to main console output routine 0264 CDD703 01111 CALL CONDUT ;60 to main console output routine 0264 CDD703 01111 CALL CONDUT ;60 to main console output routine 0264 C35F0 01114 JMP Display\$Message ;Loop until complete message output 01115 ; 01200 ;# 01200 ;# 01200 ;# 01200 ; 01205 ; input/output address (the DMA address). 01206 ; input/output address (the DMA address). 01206 ; input/output address (the DMA address). 01207 MVI A,JMP ;Get machine code for JMP 0264 C3520000 01208 STA 0000H ;Set up JMP at location 0000H 01213 2020 01209 STA 0000H ;Set up JMP at location 0000H 01210 ; 01210 ; 01211 LXI H,Warm\$Boot\$Entry ;Get BIOS vector address 0277 220100 01211 LXI H,BDD\$Entry ;Get BIOS vector address 0277 220100 01212 SHLD 60 ;Put address at location 0000H 0213 30200 01205 SHLD 60 ;Put address at location 0000H 0213 30200 01215 SHLD 60 ;Put address at location 0000H 02114 ; 0226 FB 01200 ;F 0226 FB 01200 ;F 0226 FB 01200 EIT LXI B,80H ;Set disk I/0 address to default 0239 018000 01217 LXI B,80H ;Set disk I/0 address to default 0239 018000 01217 LXI B,80H ;Set disk I/0 address to default 0239 018000 01217 LXI B,80H ;Set disk I/0 address to default 0239 018000 01217 LXI B,80H ;Set disk I/0 address to default 0239 018000 01217 LXI B,80H ;Set disk I/0 address to default 0239 018000 01217 LXI B,80H ;Set disk I/0 address to default 0239 018000 01217 LXI B,80H ;Set disk I/0 addr</pre>		00976				;Inter	rupt O (clock) enabled
 01100 ; # 01101 ; 01102 ; 01103 Display#Message: ;Displays the specified message on the console. 01103 Display#Message: ;On entry, HL points to a.stream of bytes to be ;output. A 00H-byte terminates the message. 025F 7E 01106 MOV A, M ;Get next message byte 0260 B7 01107 ORA A ;Check if terminator 0261 C8 01108 RZ ;Yes, return to caller 0262 4F 01109 MOV C, A ;Frepare for output 0263 E5 01110 PUSH H ;Set message pointer 0264 CDD703 01111 CALL CONOUT ;Go to main console output routine 0267 E1 01112 POP H ;Recover message pointer 0268 23 01113 INX H ;Horows to next byte of message 02100 ;# 01202 if# 01203 ; introver to caller from the cold or warm ; base page, and also sets the high-level disk driver's 01205 ; input/output address (the DMA address). 01206 ; 01207 MVI A, JMP ;Get machine code for JMP 01210 ; 01210 ; 012		00977				;Inter	rupt 1 (character input) enabled
01101;01102;01103Display#Message:;Displays the specified message on the console.01104;On entry, HL points to a stream of bytes to be025F 7E01106MOV0260 B701107ORA0261 C801108RZ0252 4F01109MOV0242 4F01109MOV0243 E501110PUSH0244 CDD70301111CALL0269 C35F0201113INX0269 C35F0201114JMP0269 C35F0201114JMP01203;01204;01205;01205;01206;01206;01207NN01208;01209;01203;01204;01205;01206;01205; <td< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td></td<>							
01102j01103DisplayMessage: (Dinentry, HL points to a.stream of bytes to be (Dinentry, HL point address).026501111CALLCONOUT (Set point address).026623000001200;026632000001200;026632000001200 <td< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td></td<>							
01103 01104 01105Display\$Message: :Display.the specified message on the console. :Dunput, A 00H-byte terminates the message.025F 7E 0260 B701106MOV 0107A,M:Dentry, HL points to a stream of bytes to be ;output, A 00H-byte terminates the message.0260 B7 0261 C801107ORA R7:Frepare for output :Save message pointer0262 4F 0263 E501100PUSH PUSH H:Save message pointer0264 CDD703 0264 CDD70301111 CALLCONOUT :Go to main console output routine :Recover message pointer0268 23 0268 23 02111301112 INX INX 01200POP :# :This routine is entered either from the cold or warm ; boot code. It sets up the JMP instructions in the 01201 ; boot code. It sets up the JMP instructions in the 012020266 3E53 01204:piter\$CPM: :piter\$CPM: 01205 01205:piter <td></td> <td></td> <td>;</td> <td></td> <td></td> <td></td> <td></td>			;				
01105 ; output, A 00H-byte terminates the message. 025F 7E 01107 ORA A ; Get next message byte 0260 B7 01107 ORA A ; Check if terminator 0261 C8 01108 RZ ; Yes, return to caller 0262 4F 01109 MOV C.A ; Frepare for output 0263 E5 01110 PUSH H ; Save message pointer 0264 CDD703 01111 CALL CONOUT ; Go to main console output routine 0265 C3 01114 JMP Display\$Message ; Loop until complete message output 01015 ; 01200 ;#			, Display\$Message	e:	;Displ	ays the	specified message on the console.
0256 7E 01106 MOV A,M ;Get next message byte 0260 87 01107 ORA A ;Check if terminator 0261 C8 01108 RZ ;Yes, return to caller 0263 E5 01110 PUSH H ;Save message pointer 0264 CDD703 01111 CALL CONOUT ;Go to main console output routine 0264 CDD703 01111 CALL CONOUT ;Go to main console output routine 0268 23 01113 INX H ;Move to next byte of message 0269 C35F02 01114 JMP Display\$Message ;Loop until complete message output 01200 ;# 01200 ;# 01202 Enter\$CPM: ;This routine is entered either from the cold or warm 01203 ; boot code. It sets up the JMF instructions in the 01204 ; base page, and also sets the high-level disk driver's 01205 ; input/output address (the DMA address). 01206 ; 01206 01206 ;		01104			;On en	try, HL	points to a stream of bytes to be
0260 B7 01107 ORA A ;Check if terminator 0261 C8 01108 RZ ;Yes, return to caller 0262 4F 01109 MOV C,A ;Prepare for output 0263 E5 01110 PUSH H ;Save message pointer 0264 CDR03 01111 CALL CONOUT ;Go to main console output routine 0267 E1 01112 POP H ;Recover message pointer 0268 23 01113 INX H ;Move to next byte of message 0269 C35F02 01114 UMP Display\$Message ;Loop until complete message output 01100 ;# 01200 ;# ;Diato code. It sets up the UMP instructions in the 01203 ; boot code. It sets up the UMP instructions in the ;Diato code ; ;Diato code is input/output address (the DMA address). 01204 ; base page, and also sets the high-level disk driver's ;Diato for UMP 0266 320000 01208 STA 0000H ;Set up UMP at location 0000H 0271 320500 01209 STA 00005H					;outpu		
0261 C8 01108 RZ ;Yes, return to caller 0262 4F 01109 MOV C,A ;Prepare for output 0263 E5 01110 PUSH H ;Save message pointer 0264 CDD703 01111 CALL CONUT ;Go to main console output routine 0267 E1 01112 POP H ;Reover message pointer 0268 23 01113 INX H ;Move to next byte of message 0269 C35F02 01114 JMP Display\$Message ;Loop until complete message output 01201 ;							
0262 0263 0263 02630110MOVC.A:Frepare for output0264 0264 0264 026801110PUSH PUSH H;Save message pointer0264 0268 026801111CALL POP PDP O1112CONOUT POP PDP H;Recover message pointer0268 0268 026901114JMP Display\$Message;Loop until complete message output01115 item structions in the olicol item structions in the item struction struction item struction struction item struction ite				A			
0233 ES 01110 PUSH H ;Save message pointer 0244 CDD703 01111 CALL CONOUT ;Go to main console output routine 0247 E1 01112 POP H ;Recover message pointer 0248 23 01113 INX H ;Move to next byte of message 0269 C35F02 01114 JMP Display\$Message ;Loop until complete message output 01200 ;# 01200 ; This routine is entered either from the cold or warm 01202 Enter\$CPM: ;This routine is entered either from the cold or warm ; 01203 ; input/output address (the DMA address). 01206 ; 01204 ; base page, and also sets the high-level disk driver's 01206 ; 01205 ; input/output address (the DMA address). 01206 ; 01206 ;				C.A			
0264 CDD70301111CALLCONOUT:Go to main console output routine0267 E101112POPH:Recover message pointer0268 2301113INXH:Move to next byte of message0269 C35F0201114JMPDisplay\$Message ;Loop until complete message output01115;01200;#01201;01202Enter\$CPM:01203; boot code. It sets up the JMP instructions in the01204; base page, and also sets the high-level disk driver's01205; input/output address (the DMA address).01206;01207MVI01208STA00264 2320000120801210;0265 3200000120901209STA00274 2103000121101211LXI02274 21030001211012130001H02274 2103000121401215SHLD02273 2206000121501216;02274 2103000121401213LXI0238 018000012170245;Put address at location 0005H02213CALL0239SHLD0241CALL0251SHLD0262iput address at location 0005H0274 210300012140214LXIH, BDOS\$Entry0255SHLD0266iput address at location 0005H02712206000216; <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>							
0267 E101112POPH;Recover message pointer0268 2301113INXH;Move to next byte of message0269 C35F0201114JMPDisplay\$Message ;Loop until complete message output01115;01200;#01201;01202Enter\$CPM:;This routine is entered either from the cold or warm01203; boot code. It sets up the JMP instructions in the01204; base page, and also sets the high-level disk driver's01205; input/output address (the DMA address).026C 3EC301207MVI026E 32000001208STA00264; and at location 0000H0271 32050001209STA01210;0274 2103000121101212SHLD0001H;Put address at location 0001H0277 2201000121501215SHLD0274 2106CC0121401216;02268 0180000121701217LXI0280 0180000121701218CALL0280 0180000121701219;0286 FB0122001219EI0286 FB0122001210EI0287 3A04000122101210EI0287 AD4000122101216EI0287 AD4000122101217LXI0280 FB0122001218CALL0287 AD400012210	0264 CDD703	01111					
0269 C35F0201114JMPDisplay\$Message ;Loop until complete message output01115;01200;#01201;01202Enter\$CPM:01203; boot code. It sets up the JMP instructions in the01204; base page, and also sets the high-level disk driver's01205; input/output address (the DMA address).01206;01207MVI01208STA00261 320000120901209STA00274 2103000121101210;0274 2103000121201213SHLD00274 2106CC0121401215SHLD0274 2106CC0121401216;0276 2106CC012140277 220100012150212SHLD0276 2106CC012140277 2106CC012140278 2106CC012140280 018000012170286 CB809012180287 20400012170286 FB012200287 300400012210286 FB012200287 300400012210287 500000122101216;0286 FB0122001210EI0287 3004000122101219;0287 3004000122101219;0286 FB0122001220EI0287 300400012210120EI0287 300400012210120	0267 E1						
 01115 ; 01200 ;# 01201 ; 01202 Enter\$CPM: ;This routine is entered either from the cold or warm 01203 ; boot code. It sets up the JMP instructions in the 01204 ; base page, and also sets the high-level disk driver's 01205 ; input/output address (the DMA address). 0266 3EC3 01207 MVI A, JMP ; Get machine code for JMP 026E 320000 01208 STA 0000H ; Set up JMP at location 0000H 0271 320500 01209 STA 0005H ; and at location 0005H 01210 ; 0274 210300 01211 LXI H,Warm\$Boot\$Entry ;Get BIOS vector address 0277 220100 01212 SHLD 0001H ;Put address at location 0001H 01213 0277 220400 01215 SHLD 6 ;Put address at location 0005H 01216 ; 0278 2106CC 01214 LXI H,BDOS\$Entry ;Get BDOS entry point address 0277 220400 01215 SHLD 6 ;Put address at location 0005H 01216 ; 0280 018000 01217 LXI B,S0H ;Set disk I/O address to default 0283 CDA809 01218 CALL SETDMA ;Use normal BIOS routine 01219 ; 0286 FB 01220 EI ;Ensure interrupts are enabled 0287 3A0400 01211 LDA Default\$Disk ;Handover current default disk to 	0268 23				* M	;Move	to next byte of message
01200;#01201;01202Enter\$CPM:01203; boot code. It sets up the JMP instructions in the01204; base page, and also sets the high-level disk driver's01205; input/output address (the DMA address).01206;01207MVI01208STA0000H; Set up JMP at location 0000H027421030001210;027421030001210;027421030001211LXIH,Warm\$Boot\$Entry027421040001212SHLD0001H; Put address at location 0005H027421040001212SHLD00214LXIH,BDO\$Entry; Get BDO\$ entry point address0270012150274SHLD027410400027421040001212SHLD027410400027421040001212SHLD02741021402741040002752040001215SHLD6; Put address at location 0005H0280012171218CALL02800121802801218028012190286FB01220EI0286FB01220EI0286FB01220EI02872040001211LDA0280	0269 C35F02			Display	pmessage	e ;Loop	untii complete message output
01201;01202Enter\$CPM:01203; boot code. It sets up the JMP instructions in the01204; base page, and also sets the high-level disk driver's01205; input/output address (the DMA address).026C 3EC30120701208STA00264; Get machine code for JMP026501209026632000001209STA0000H; set up JMP at location 0000H027132050001210;027421030001211LXIH,Warm\$Boot\$Entry027722010001212SHLD0001H02742106CC0121302742106CC01214LXIH,BDD\$\$Entry; Get BDD\$ certor address0270021210274028001800001217LXIB,80H; Set disk I/0 address to default02830284702040001217LXIB,80H; Set disk I/0 address to default0286028001209CA8400001217LXIB,80H; Set disk I/0 address to default0287028040001217LXIB,80H; Set disk I/0 address to default02860120901209CA8400001210LXI							
01202Enter\$CPM:;This routine is entered either from the cold or warm ; boot code. It sets up the JMP instructions in the 0120401204; boot code. It sets up the JMP instructions in the 0120501205; input/output address (the DMA address).01206;01206;026C 3EC301207MVI026E 32000001208STA0000H; Set machine code for JMP026E 32000001209STA00271 32050001210;01210;0277 22010001212SHLD01211LXIH,Warm\$Boot\$Entry0277 22010001215SHLD01215SHLD60270 22060001215SHLD0271 32050001215SHLD0277 22010001212SHLD012130001H0270 220600012150280 01800001217LXI0280 01800001217LXI0280 01800001217LXI0280 61800001217LXI0280 61800001217LXI0280 678001220EI0280 678001220EI0280 678001220EI0280 678001220EI0280 678001220EI0280 678001221LDA0280 678001221LDA0280 678001221LDA0280 678001220EI0280 678001220EI0280 78040001221LDA			:				
01203: boot code. It sets up the JMP instructions in the oli20401204: base page, and also sets the high-level disk driver's oli20501205: input/output address (the DMA address).026C 3EC301207026E 3200000120801209STA0000H: Set up JMP at location 0000H 012100274 2103000121101210:0274 2103000121101210:0274 2103000121101210:0274 2104000121101212SHLD0001H: Put address at location 0001H0274 2106CC0121401213LXI0274 2106CC0121401213.0274 2106CC0121401214LXIH, BD0S\$Entry:Get BD0S entry point address0270 20260001216:0280 0180000121701219:<			Enter\$CPM:	;This r	outine	is enter	ed either from the cold or warm
01205 ; input/output address (the DMA address). 01206 ; 01207 MVI 026C 3EC3 01207 01208 STA 0000H ; Set up JMP at location 0000H 0271 320500 01209 01210 ; 0277 220100 01211 LXI H,Warm\$Boot\$Entry 0277 220100 01212 01213 SHLD 0001H ; Put address at location 0001H 01213 SHLD 00271 220100 01212 01213 SHLD 0277 220100 01212 01213 SHLD 0277 220600 01215 01215 SHLD 0270 220600 01215 01216; ;Put address at location 0005H 0280 018000 01217 LXI B,80H ;Set disk I/O address to default 0283 CDA809 01218 CALL SETDMA 0286 FB 01220 EI ;Ensure interrupts are enabled 0287 340400 01212 LDA Default\$Disk ;Handover curre		01203		; boot	code. I	t sets u	p the JMP instructions in the
01206 ; MVI A, JMP ; Get machine code for JMP 0266 320000 01207 MVI A, JMP ; Get machine code for JMP 0265 320000 01208 STA 0000H ; Set up JMP at location 0000H 0271 320500 01209 STA 0005H ; and at location 0005H 0274 210300 01211 LXI H, Warm\$Boot\$Entry ; Get BIOS vector address 0277 220100 01212 SHLD 0001H ; Put address at location 0001H 0274 2106CC 01214 LXI H, BDO\$Entry ; Get BDO\$ entry point address 0270 220600 01215 SHLD 6 ; Put address at location 0005H 0270 220600 01215 SHLD 6 ; Put address to default 0280 018000 01217 LXI B,80H ; Set disk I/0 address to default 0283 CDA809 01218 CALL SETDMA ; Use normal BIOS routine 0287 204000 01221 LDA Default*Disk ; Handover current default disk to				; base	page, a	nd also	sets the high-level disk driver's
026C 3EC3 01207 MVI A, JMP ;Get machine code for JMP 026E 320000 01208 STA 0000H ;Set up JMP at location 0000H 0271 320500 01209 STA 0005H ; and at location 0005H 0274 210300 01210 ;				; input	/output	address	(the DMA address).
026E 320000 01208 STA 0000H ;Set up JMP at location 0000H 0271 320500 01209 STA 0005H ; and at location 0005H 01210 ;	0040 9509		7 MU T			·Gat m	achine code for IMP
0271 320500 01209 STA 0005H ; and at location 0005H 0274 210300 01210 ; H,Warm\$Boot\$Entry ;Get BIOS vector address 0277 220100 01212 SHLD 0001H ;Put address at location 0005H 0274 210300 01212 SHLD 0001H ;Put address at location 0001H 0274 2106CC 01214 LXI H,BD0\$Entry ;Get BD0S entry point address 0270 20600 01215 SHLD 6 ;Put address at location 0005H 01216 ;							
01210 ; 0274 210300 01211 LXI H,Warm\$Boot\$Entry ;Get BIOS vector address 0277 220100 01212 SHLD 0001H ;Put address at location 0001H 01213 01213 01213 01213 01213 0277 220600 01214 LXI H,BDOS\$Entry ;Get BDOS entry point address 0270 220600 01215 SHLD 6 ;Put address at location 0005H 01216 ;						; and	at location 0005H
0274 210300 01211 LXI H,Warm\$Boot\$Entry :Get BIOS vector address 0277 220100 01212 SHLD 0001H :Put address at location 0001H 01213							
01213 01213 027A 2106CC 01214 LXI H,BDOS\$Entry ;Get BDOS entry point address 027D 220600 01215 SHLD 6 ;Put address at location 0005H 01216 ; 01216 ; 0280 018000 01217 LXI B,80H ;Set disk I/O address to default 0283 CDA809 01218 CALL SETDMA ;Use normal BIOS routine 01219 ; ; ;Ensure interrupts are enabled 0287 50400 01221 LDA Default\$Disk ;Handover current default disk to		01211	LXI		Boot \$En		
027A 2106CC 01214 LXI H, BDOS\$Entry :Get BDOS entry point address 027D 220600 01215 SHLD 6 :Put address at location 0005H 01216 0280 018000 01217 LXI B,80H ;Set disk I/O address to default 0283 CDA809 01218 CALL SETDMA ;Use normal BIOS routine 01219 0287 204000 01220 EI . . 0287 3A0400 01221 LDA Default\$Disk ;Handover current default disk to	0277 220100		SHLD	0001H		;Put a	ddress at location 0001H
027D 220600 01215 SHLD 6 ;Put address at location 0005H 01216 ; 0 <td< td=""><td>0074 010/00</td><td></td><td></td><td>u phoce</td><td>Entry</td><td>IGet P</td><td>1005 entry point address</td></td<>	0074 010/00			u phoce	Entry	IGet P	1005 entry point address
01216 ; 0280 018000 01217 LXI B,80H ;Set disk I/O address to default 0283 CDA809 01218 CALL SETDMA ;Use normal BIOS routine 01219 ; 0286 FB 01220 EI ;Ensure interrupts are enabled 0287 3A0400 01221 LDA Default\$Disk ;Handover current default disk to						; oet B	ddress at location 0005H
0280 018000 01217 LXI B,80H ;Set disk I/O address to default 0283 CDA809 01218 CALL SETDMA ;Use normal BIOS routine 01219 ;	02/D 220000			0		, u. a	
0283 CDA809 01218 CALL SETDMA ;Use normal BIOS routine 01219 ; 0286 FB 01220 EI ;Ensure interrupts are enabled 0287 3A0400 01221 LDA Default\$Disk ;Handover current default disk to	0280 018000			B,80H		;Set d	iisk I/O address to default
01219 ; 0286 FB 01220 EI ;Ensure interrupts are enabled 0287 3A0400 01221 LDA Default\$Disk ;Handover current default disk to		01218				;Use n	ormal BIOS rouține
0287 3A0400 01221 LDA Default\$Disk ;Handover current default disk to							
0287 4F 01222 MOV C,A ; console command processor	0286 FB			Dofault	#Dick	;Ensur	e interrupts are enabled wer current default disk to
					+DISK	; cons	ole command processor

Figure 8-10. (Continued)

[······		
028B C300C4	01223	UMP CCP\$Ent	try	;Transfer to CCP
	01224	;		
	01300	;#		
	01301	;		
	01302	; Device table eq	quates	
	01303	; The drivers use	a devi	ce table for each
1	01304			ervice. The equates that follow
	01305			various fields within the
	01306	; device table.		
	01307	•		
1	01308		Port n	umbers and status bits
0000 =	01309	DT\$Status\$Port	EQU	0 ;Device status port number
0001 =	01310	DT\$Data\$Port	EQU	DT\$Status\$Port+1
	01311	Dividentian	200	;Device data port number
0002 =	01312	DT\$Output\$Ready	EQU	DT\$DataPort+1
****	01313	privatpatineady	200	Output ready status mask
0003 =	01314	DT\$Input\$Ready	EQU	DT\$Output\$Ready+1
0000	01315	Diffingat fileda)	Lac	;Input ready status mask
0004 =	01316	DT\$DTR\$Ready	EQU	DT\$Input\$Ready+1
0004 -	01317	DIPDINPNEADY	Lao	;DTR ready to send mask
0005 =	01318	DT\$Reset\$Int\$Port	EQU	DT\$DTR\$Ready+1
	01318	2, 4116361411119FUF1	680	
	01319			;Port number used to reset an
0006 =	01320	DT\$Reset\$Int\$Value	EQU	; interrupt
0008 -		nisuezersturs/sing	EWO	DT\$Reset\$Int\$Port+1
0007 -	01322		Four	;Value output to reset interrupt
0007 =	01323	DT\$Detect\$Error\$Port	EQU	DT\$Reset\$Int\$Value+1
	01324			;Port number for detecting error
0008 =	01325	DT\$Detect\$Error\$Value	EQU	DT\$Detect\$Error\$Port+1
	01326			;Mask for detecting error (parity etc.)
0009 =	01327	DT\$Reset\$Error\$Port	EQU	DT\$Detect\$Error\$Value+1
	01328			;Output to port to reset error
000A =	01329	DT\$Reset\$Error\$Value	EQU	DT\$Reset\$Error\$Port+1
	01330			;Value to output to reset error
000B =	01331	DT\$RTS\$Control\$Port	EQU	DT\$Reset\$Error\$Value+1
	01332			Control port for lowering RTS
000C =	01333	DT\$Drop\$RTS\$Value	EQU	DT\$RTS\$Control\$Port+1
	01334			;Value, when output, to drop RTS
000D =	01335	DT\$Raise\$RTS\$Value	EQU	DT\$Drop\$RTS\$Value+1
	01336			;Value; when output, to raise RTS
	01337	;		
	01338	; Device	logical	status (incl. protocols)
000E =	01339	DT\$Status	EQU	DT\$Raise\$RTS\$Value+1
	01340			;Status bits
0001 =	01341	DT\$Output\$Suspend	EQU	0000\$0001B ;Output suspended pending
••••	01342			; protocol action
0002 =	01343	DT\$Input\$Suspend	EQU	0000\$0010B ;Input suspended until
	01344	a. sengationappend		; buffer empties
0004 =	01345	DT\$Output\$DTR	EQU	0000\$0100B ;Output uses DTR-high-to-send
0004 =	01346	DT\$Output\$Xon	EQU	0000\$1000B ;Output uses XON/XOFF
0010 =	01348	DT\$Output\$Etx	EQU	0001\$0000B ;Output uses ETX/ACK
0020 =	01348	DT\$Output\$Timeout	EQU	0010\$0000B ;Output uses timeout
0020 =	01348	DT\$Input\$RTS	EQU	0100\$0000B ;Input uses RTS-high-to-receive
0040 =	01349	DT\$Input\$Xon	EQU	1000\$0000B ;Input uses XIS-Nigh-to-receive
0080 -	01350	DI #INPUT#KON	600	TOOD#OOOD ;INPUT USES XON/XOFF
000F =	01351	; DT\$Status\$2	EQU	DT\$Status+1 ;Secondary status byte
0001 =	01353	DT\$Fake\$Typeahead	EQU	0000\$0001B ;Requests Input\$Status to
	01354			; return "Data Ready" when
	01355			; control dharacters are in
	01356			; input buffer
	01357			
0010 =	01358	DT\$Etx\$Count	EQU	DT\$Status\$2+1
	01359			;No. of chars.sent in Etx protocol
0012 =	01360	DT\$Etx\$Message\$Length	EQU	DT\$Etx\$Count+2
	01361			;Specified message length
	01362	;	_ .	
	01363	;		buffer values
	01364	DT\$Buffer\$Base	EQU	DT\$Etx\$Message\$Length+2
0014 =				;Address of Input buffer
	01365		EQU	DT\$Buffer\$Base+2
001 4 = 0016 =	01366	DT\$Put\$Offset		
0016 =	01366 01367			;Offset for putting chars. into buffer
	01366 01367 01368	DT\$Put\$Offset DT\$Get\$Offset	EQU	DT\$Put\$Offset+1
0016 = 0017 =	01366 01367			DT\$Put\$Offset+1 ;Offset for getting chars. from buffer
0016 =	01366 01367 01368		EQU	DT\$Put\$Offset+1 ;Offset for getting chars. from buffer
0016 = 0017 =	01366 01367 01368 01369	DT\$Get\$Offset		DT\$Put\$Offset+1 ;Offset for getting chars. from buffer DT\$Get\$Offset+1
0016 = 0017 =	01366 01367 01368 01369 01370	DT\$Get\$Offset		DT\$Put\$Offset+1 ;Offset for getting chars. from buffer DT\$Get\$Offset+1 ;Length of buffer - 1
0016 = 0017 =	01366 01367 01368 01369 01370 01371	DT\$Get\$Offset		DT\$Put\$Offset+1 ;Offset for getting chars. from buffer DT\$Get\$Offset+1

	01374					;This mask then becomes:
	01375					; 32 -> 31 (0001\$1111B)
	01376					; 64 -> 63 (0011\$1111B)
	01377					; 128 -> 127 (0111\$1111B)
	01378					;After the get/put offset has been
	01379					; incremented, it is ANDed with the mask
	01380					; to reset it to zero when the end of
0019 =	01381 01382	DTACH	acter\$Co		EQU	; the buffer has been reached DT\$Buffer\$Length\$Mask+1
0019 =	01382	DIDUNAR	acteraco	unt /	EQU	Count of the number of characters
	01385					; currently in the buffer
001A =	01385	DT\$Stop	\$Input\$C	ount	EQU	DT\$Character\$Count+1
	01386	2.12.00	+======			;Stop input when the count reaches
	01387					; this value
001B =	01388	DT\$Resu	me\$Input	\$Count	EQU	DT\$Stop\$Input\$Count+1
	01389					Resume input when the count reaches;
	01390					; this value
001C =	01391	DT\$Cont	ro1\$Coun	t	EQU	DT\$Resume\$Input\$Count+1
	01392					;Count of the number of control
	01393				-	; characters in the buffer
001D =	01394	DT \$ Func	tion\$Del	ау	EQU	DT\$Control\$Count+1
	01395					Number of clock ticks to delay to
	01396					; allow all characters after function
0045	01397	DT#1-14			EQU	; key lead-in to arrive
001E =	01398 01399	DI⊅Init	ialize\$S	iream	EQU	DT\$Function\$Delay+1 ;Address of byte stream necessary to
	01399					; initialize this device
	01400					, INICIDIZE CHID DEVICE
	01500	;#				
	01501	;				
	01502	;	Device	tables		
	01503	;				
	01504	DT\$0:				
028E 81	01505	-	DB		us\$Port	;Status port (8251A chip)
028F 80	01506		DB	DO\$Data		;Data port
0290 01	01507		DB		t\$Ready	;Output data ready
0291 02	01508		DB	D\$Input		;Input data ready
0292 80	01509		DB	D\$DTR\$H		;DTR ready to send
0293 D8	01510		DB DB	IC\$0CW2 IC\$E0I	sport	;Reset interrupt port (00H is an junused port) ;Reset interrupt value (nonspecific EOI)
0294 20 0295 81	01511 01512		DB		us\$Port	;Detect error port
0296 38	01513		DB	D\$Error		;Mask: framing, overrun, parity errors
0297 83	01514		DB			;Reset error port
0298 37	01515		DB	D\$Error	\$Reset	;Reset error: RTS high, reset, Tx/Rx enable
0299 83	01516		DB	DO\$Comm	and\$Port	;Drop/raise RTS port
029A 07	01517		DB	D\$Drop\$	RTS	;Drop RTS Value (keep Tx & Rx enabled)
029B 27	01518		DB	D\$Raise	\$RTS	;Raise RTS value (keep Tx & Rx enabled)
029C C0	01519		DB	DT\$Inpu	it\$Xon + 1	DT\$Input\$RTS ;Protocol and status
029D 00	01520					
			DB	0		;Status #2
029E 0004	01521		DW	0 1024		;Status #2 ;Etx/Ack message count
02A0 0004	01521 01522		DW DW	0 1024 1024		:Status #2 ;Etx/Ack message count ;Etx/Ack message length
02A0 0004 02A2 2422	01521 01522 01523		DW DW DW	0 1024 1024 D0\$Buff		;Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer
02A0 0004 02A2 2422 02A4 00	01521 01522 01523 01524		DW DW DW DB	0 1024 1024		;Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer
02A0 0004 02A2 2422 02A4 00 02A5 00	01521 01522 01523 01524 01525		DW DW DW	0 1024 1024 D0\$Buff 0 0	er	;Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F	01521 01522 01523 01524 01525 01526		DW DW DW DB DB DB	0 1024 1024 D0\$Buff 0 0	er	;Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer -1 ;Buffer length mask
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00	01521 01522 01523 01524 01525 01526 01527		DW DW DW DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff	er er\$Lengt er\$Lengt	;Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F	01521 01522 01523 01524 01525 01526 01527 01528 01529		DW DW DW DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff	er er\$Lengt er\$Lengt	;Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer -1 ;Buffer length mask
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02AA 00	01521 01522 01523 01524 01525 01526 01527 01528 01529 01530		DW DW DW DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff D0\$Buff 0	er er\$Lengt er\$Lengt	:Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value h / 2 ;Resume input when count hits this value ;Count of control characters in buffer
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10	01521 01522 01523 01524 01525 01526 01526 01527 01528 01529 01530 01531		DW DW DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff D0\$Buff	er er\$Lengt er\$Lengt	<pre>;Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value n / 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 00 02AB 06	01521 01522 01523 01524 01525 01526 01527 01528 01529 01530 01531 01532		DW DW DW DB DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff D0\$Buff 0 6	er er\$Lengt er\$Lengt er\$Lengt	<pre>Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Get offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms)</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02AA 00	01521 01522 01523 01524 01525 01526 01527 01528 01529 01530 01531 01532		DW DW DW DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff D0\$Buff 0 6	er er\$Lengt er\$Lengt	<pre>Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms)</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 00 02AB 06	01521 01522 01523 01524 01525 01526 01527 01528 01529 01530 01531 01533 01533	;	DW DW DW DB DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff D0\$Buff 0 6	er er\$Lengt er\$Lengt er\$Lengt	<pre>Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Get offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms)</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A8 10 02A8 10 02A8 00 02A8 00 02A8 00	01521 01522 01523 01525 01525 01526 01527 01528 01529 01530 01531 01532 01533	; DT\$1:	DW DW DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff 0 6 D0\$Init	er er\$Lengt er\$Lengt er\$Lengt .ialize\$S [.]	<pre>:Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer n - 1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value n / 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 06 02AC 8400 02AE 85	01521 01522 01523 01525 01525 01526 01527 01528 01530 01531 01533 01533 01533	; DT\$1:	DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff 0 6 D0\$Init D1\$Stat	er er\$LengtH er\$LengtH er\$LengtH .ialize\$S: .us\$Port	<pre>;Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value n / 2 ;Resume input when count hits this value iCount of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip)</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 00 02A8 00 02A8 8400 02AE 85 02AF 84	01521 01522 01523 01524 01525 01526 01526 01528 01529 01530 01531 01532 01533 01534 01535 01534	; DT\$1:	DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff D0\$Buff 0 6 D0\$Init D1\$Stat D1\$Data	er er\$LengtH er\$LengtH er\$LengtH .ialize\$& :u\$\$Port	<pre>:Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Get offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value n/ 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip) ;Data port</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 00 02A8 06 02A8 85 02AF 84 02B0 01	01521 01522 01523 01525 01525 01526 01527 01528 01529 01530 01531 01532 01533 01534 01535 01535 01535	; DT\$1:	DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 204Buff 0 0 00\$Buff 0 0 00\$Buff 0 6 0 0 6 D0\$Buff 0 6 D0\$Buff 0 0 5 D0\$Buff 0 0 5 D0\$Buff 0 0 5 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff D0\$Buff D0\$B	er er\$Lengt er\$Lengt ialize\$S us\$Port u\$Port u\$Ready	<pre>:Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n -5 ;Stop input when count hits this value n / 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip) ;Data port ;Output data ready</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 06 02A8 06 02AC 8400 02AE 85 02AF 84 02B1 02	01521 01522 01523 01524 01525 01526 01526 01528 01528 01530 01531 01532 01533 01534 01535 01534 01535 01538 01538	; DT\$1:	DW DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff 0 6 D0\$Init D1\$Stat D1\$Data D\$Uutpu D5Input	er er\$Length er\$Length er\$Length ialize\$S us\$Port u\$Port u\$Ready	<pre>:Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer n - 1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value n / 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip) ;Data port ;Output data ready ;InTR ready to send</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 00 02A8 00 02A8 8400 02AC 8400 02AF 84 02B0 01 02B1 02 02B2 80	01521 01522 01523 01524 01525 01526 01527 01528 01529 01530 01532 01532 01533 01534 01535 01534 01537 01538 01539 01540	; DT\$1:	DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 204Buff 0 0 00\$Buff 0 0 00\$Buff 0 6 0 0 6 D0\$Buff 0 6 D0\$Buff 0 0 5 D0\$Buff 0 0 5 D0\$Buff 0 0 5 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff 0 0 D0\$Buff D0\$Buff D0\$B	er er\$Lengt er\$Lengt ialize\$S us\$Port i\$Port i\$Ready \$Ready igh	<pre>:Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer n - 1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value n / 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip) ;Data port ;Output data ready ;InTR ready to send</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 06 02AC 8400 02AE 85 02AF 84 02B1 02	01521 01522 01523 01524 01525 01526 01526 01528 01528 01530 01531 01532 01533 01534 01535 01534 01535 01538 01538	; DT\$1:	DW DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff 0 6 D0\$Buff 0 6 D0\$Init D1\$Stat D1\$Stat D1\$D1\$Pt D1\$D1\$Pt D0\$D1\$Pt D1\$D1\$Pt D1\$D1\$Pt D0\$D1\$Pt D1\$D2 D1\$D	er er\$Lengt er\$Lengt ialize\$S us\$Port i\$Port i\$Ready \$Ready igh	<pre>:Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Put offset into buffer ;Get offset into buffer ;Count of characters in buffer n - 5 ;Stop input when count hits this value n / 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip) ;Data port ;Output data ready ;InTR ready to send</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 06 02A8 06 02A8 85 02AF 84 02B0 01 02B1 02 02B2 80 02B3 08	01521 01522 01523 01525 01525 01526 01527 01528 01529 01530 01531 01532 01533 01534 01538 01538 01538 01538 01538	; ⊡T\$1:	DW DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 00\$Buff 0 00\$Buff 0 00\$Buff 0 6 00\$Init D0\$Init D1\$Stat D1\$Data D5Input D5Input D5Input C\$EOI	er er\$Lengt er\$Lengt ialize\$S us\$Port i\$Port i\$Ready \$Ready igh	<pre>:Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Get offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n -5 ;Stop input when count hits this value n / 2 ;Resume input when count hits this value / 2 ;Resume input when count hits this value i/ 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip) ;Data port ;Output data ready ;Input data ready ;DTR ready to send ;Rest interrupt port (00H is an unused port)</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02AA 00 02AB 06 02AC 8400 02AC 8400 02AE 85 02AF 84 02B0 01 02B1 02 02B2 80 02B3 D8 02B4 20	01521 01522 01523 01524 01525 01526 01526 01528 01529 01530 01530 01532 01533 01534 01534 01534 01538 01538 01539 01540	; DT\$1:	DW DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 00\$Buff 0 00\$Buff 0 00\$Buff 0 6 00\$Init D0\$Init D1\$Stat D1\$Data D5Input D5Input D5Input C\$EOI	er er\$Lengt er\$Lengt ialize\$S us\$Port it\$Ready \$Ready igh 2\$Port :us\$Port	<pre>Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Get offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value n/ 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port ;Output data ready ;Input data ready ;Reset interrupt value (nonspecific EOI)</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 06 02A8 8400 02A8 85 02AC 8400 02A8 85 02A6 84 02B0 01 02B1 02 02B2 80 02B3 D8 02B4 20 02B5 85 02B6 38 02B7 87	01521 01522 01523 01524 01525 01526 01527 01528 01528 01530 01530 01532 01533 01534 01535 01534 01538 01538 01538 01538 01539 01540 01541 01544 01544	; DT\$1:	DW DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 D0\$Buff 0 D0\$Buff 0 D0\$Buff 0 6 D0\$Init D1\$Stat D\$Init D\$Init D\$TR\$H IC\$EOI D1\$Stat D\$DTR\$H IC\$EOI D1\$Stat D\$DTR\$H	er er\$LengtH er\$LengtH .ialize\$S us\$Port ut\$Ready High \$Port us\$Port us\$Port	<pre>Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Get offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip) ;Data port ;Output data ready ;Input data ready ;InTR ready to send ;Reset interrupt port (00H is an unused port) ;Detect error port ;Mask: framing, overrun, parity errors ;Reset error port</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 00 02A8 00 02A8 85 02AC 8400 02AE 85 02AF 84 02B0 01 02B1 02 02B3 08 02B3 08 02B3 08 02B3 85 02B4 38 02B7 87 02B6 38	01521 01522 01523 01525 01525 01526 01527 01528 01529 01530 01531 01532 01533 01534 01538 01538 01538 01538 01538 01538 01544 01544	; DT\$1:	DW DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 1024 0 0 00\$Buff 0 0 00\$Buff 0 6 0 0 6 0 0 6 0 0 1 \$Da\$Buff 0 0 6 0 0 1 \$Da\$Buff 0 0 6 0 0 1 \$Da\$Buff 0 0 1 \$Da\$Buff 0 0 0 1 0 0 5 0 8 0 1 0 0 0 0 0 8 0 1 0 0 0 0 0 1 0 0 0 0	er er\$Lengt er\$Lengt ialize\$& us\$Port u\$Port i\$Ready igh sPort us\$Port mand\$Port	<pre>Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Get offset into buffer ;Get offset into buffer ;Get of characters in buffer n - 1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value n / 2 ;Resume input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip) ;Data port ;Output data ready ;Input data ready ;Input data ready ;Reset interrupt port (00H is an unused port) ;Reset interrupt value (nonspecific EOI) ;Petect error port ;Reset error; RTS high, reset, Tx/Rx enable</pre>
02A0 0004 02A2 2422 02A4 00 02A5 00 02A6 1F 02A7 00 02A8 1B 02A9 10 02A8 00 02A8 00 02A8 8400 02AC 8400 02AF 84 02B0 01 02B1 02 02B2 80 02B3 D8 02B4 20 02B5 85 02B6 38 02B7 87	01521 01522 01523 01524 01525 01526 01527 01528 01528 01530 01530 01532 01533 01534 01535 01534 01538 01538 01538 01538 01539 01540 01541 01544 01544	; DT\$1:	DW DW DW DB DB DB DB DB DB DB DB DB DB DB DB DB	0 1024 1024 1024 0 0 00\$Buff 0 0 00\$Buff 0 6 0 0 6 0 0 6 0 0 1 \$Da\$Buff 0 0 6 0 0 1 \$Da\$Buff 0 0 6 0 0 1 \$Da\$Buff 0 0 1 \$Da\$Buff 0 0 0 1 0 0 5 0 8 0 1 0 0 0 0 0 8 0 1 0 0 0 0 0 1 0 0 0 0	er er\$Lengt! er\$Lengt! ialize\$S: us\$Port is\$Port is\$Ready igh \$Port us\$Port s\$Port and\$Port *Reset nand\$Port	<pre>Status #2 ;Etx/Ack message count ;Etx/Ack message length ;Input buffer ;Get offset into buffer ;Get offset into buffer n -1 ;Buffer length mask ;Count of characters in buffer n - 5 ;Stop input when count hits this value ;Count of control characters in buffer ;Number of 16.66ms ticks to allow function ; key sequence to arrive (approx. 90ms) tream ;Address of initialization stream ;Status port (8251A chip) ;Data port ;Output data ready ;Input data ready ;InTR ready to send ;Reset interrupt port (00H is an unused port) ;Detect error port ;Mask: framing, overrun, parity errors ;Reset error port</pre>

02BB 27 01549 DB D\$Raise\$RTS ;Raise RTS value (keep Tx & Rx enabled) 02BC 00 01550 DB DT\$Input\$Xon + DT\$Input\$RTS ;Protocol and status 02BD 00 01551 DB 0 ;Status #2 02BE 0004 01552 DW 1024 ;Etx/Ack message count 02C0 0004 01553 DW 1024 ;Etx/Ack message length	
O2BC C0 01550 DB DT\$Input\$Xon + DT\$Input\$RTS ;Protocol and status 02BD 00 01551 DB 0 ;Status #2 02BE 0004 01552 DW 1024 ;Etx/Ack message count	
02BD 00 01551 DB 0 ; Status #2 02BE 0004 01552 DW 1024 ; Etx/Ack message count	
02BE 0004 01552 DW 1024 ;Etx/Ack message count	
j O2CO OOO4 O1553 DW 1024 :Ftx/Ack message length	
02C2 4422 01554 DW D1\$Buffer ;Input buffer	
02C4 00 01555 DB 0 ;Put offset into buffer	
02C5 00 01556 DB 0 ;Get offset into buffer	
02C7 00 01558 DB 0 ;Count of characters in buffer	
02C8 1B 01559 DB D1\$Buffer\$Length - 5 ;Stop input when count hits this va	alue
02C9 10 01560 DB D1\$Buffer\$Length / 2 ;Resume input when count hits this	value
02CA 00 01561 DB 0 ;Count of control characters in buffer	
02CB 06 01562 DB 6 ;Number of 16.66ms ticks to allow functi	ion
0200 06 01362 DB 6 9Muller 01 16.66ms ticks to allow function	100
01563 ; key sequence to arrive (approx. 90ms)	
02CC 9400 01564 DW D1\$Initialize\$Stream ;Address of initialization stream	aw.
01565 ;	
01566 ;	
01567 DT\$2:	
02CE 89 01568 DB D2\$Status\$Port ;Status port (8251A chip)	
02CF 88 01569 DB D2\$Data\$Port ;Data port	
02D0 01 01570 DB D\$Output\$Ready ;Output data ready	
02D1 02 01571 DB D\$Input\$Ready ;Input data ready	
02D2 80 01572 DB D\$DTR\$High ;DTR ready to send	
02D3 D8 01573 DB IC\$0CW2\$Port ; Reset interrupt port (00H is an unused	port)
	•
02D5 89 01575 DB D2\$Status\$Port ;Detect error port	
02D6 38 01576 DB D\$Error ;Mask: framing, overrun, parity errors	
02D7 8B 01577 DB D2\$Command\$Port ;Reset error port	
02D8 37 01578 DB D\$Error\$Reset ;Reset error: RTS high, reset, Tx/Rx ena	able
02D9 8B 01579 DB D2=Command =Fort ; Drop/raise RTS port	
02DB 27 01581 DB D\$Raise\$RTS ;Raise RTS value (keep Tx & Rx enabled)	
02DC C0 01582 DB DT\$Input\$Xon + DT\$Input\$RTS ;Protocol and status	
02DD 00 01583 DB 0 ;Status #2	
02DE 0004 01584 DW 1024 ;Etx/Ack message count	
02E0 0004 01585 DW 1024 ;Etx/Ack message length	
02E4 00 01587 DB 0 ;Put offset into buffer	
02E5 00 01588 DB 0 ;Get offset into buffer	
02E6 1F 01589 DB D2\$Buffer\$Length ~1 ;Buffer length mask	
02E7 00 01590 DB 0 ;Count of characters in buffer	
02E8 1B 01591 DB D2\$Buffer\$Length - 5 ;Stop input when count hits this ve	ماراد
0259 10 01592 DB D248Uffer8Length / 2;Resume input when count hits this	
	value
02EA 00 01593 DB 0 ;Count of control characters in buffer	
02EB 06 01594 DB 6 ;Number of 16.66ms ticks to allow functi	ion
01595 ; Key sequence to arrive (approx. 90ms)	
O2EC A400 01596 DW D2\$Initialize\$Stream ;Address of initialization strea	
0.000	
01701 ; General character I/O device initialization	
01702 ;	
01703 ; This routine will be called from the main CP/M	
01704 ; initialization code.	
01705 ;	
01706 ; It makes repeated calls to the specific character I/O	
01708 ;	
01709 General\$CIO\$Initialization:	
02EE AF 01710 XRA A ;Set device number (used to access the	
01711 ; table of device table addresses in th	1e
01712 ; configuration block)	-
02EF 4F 01713 MOV C,A ;Match to externally CALLable interface	
01714 GCI\$Next\$Device:	
02F0 CDFA02 01715 CALL Specific\$CIO\$Initialization ;Initialize the device	
02F3 3C 01716 INR A ;Move to next device	
02F4 FE10 01717 CPI 16 ;Check if all possible devices (0 - 15)	
02F6 C8 01718 RZ ; have been initialized	
01720 ;	
01800 ;#	
01801 ;	
01802 ; Specific character I/O initialization	
01804 ; This routine outputs the specified byte values to the specified	
01805 ; ports as controlled by the initialization streams in the	

	01807	; these	streams. The dev	ice table itself is selected according
	01808	; to the	device NUMBER -	- this is an entry parameter for this
	01809	; routine		
	01810	; Thiš ro		alled either from the general device
	01811			above, or directly by a BIOS call from
	01812	; a syste	em utility execu	ting in the TPA.
[01813	;		
	01814	; Entryp	parameters	
	01815	1	C = device num	hav
	01816 01817	;	C = device num	Der
	01817	Frit D	arameters	
	01819	; EXICPS	ar ameter s	
	01820	;	A = Device num	ber (preserved)
	01821	;		
	01822	;======================================		
	01823	Specific\$CIO\$I		;<=== BIOS entry point (private)
	01824	;======================================		
02FA 79	01825	MOV	A,C	;Get device number
02FB F5	01826	PUSH	PSW	;Preserve device number
02FC 87	01827	ADD	A	;Make device number into word pointer
02FD 4F 02FE 0600	01828 01829	MOV MVI	C,A B,O	;Make into a word
0300 216400	01829			ble\$Addresses ;Get table base
0303 09	01830	DAD	B	;HL -> device table address
0304 5E	01832	MOV	Ē,M	;Get LS byte
0305 23	01833	INX	H H	,
0306 56	01834	MOV	D, M	;Get MS byte: DE -> device table
1	01835			
0307 7A	01836	MOV	A, D	;Check if device table address = 0
0308 B3	01837	ORA	E	
0309 CA1703	01838	JZ	SCI\$Exit	;Yes, device table nonexistent
	01839			/
030C 211E00 030F 19	01840	LXI	H,DT\$Initializ	
030F 19	01841	DAD	D	;HL -> initialization stream address
0310 5E	01842	MOV	E,M	;Get LS byte
0311 23	01843 01844	INX MOV	н р.м	;Get MS byte
0313 EB	01845	XCHG	10,11	;HL -> initialization stream itself
0314 CD1903	01846	CALL	Output\$Byte\$St	
	01847	,		; ports
	01848	;		
	01849	SCI\$Exit:		
0317 F1	01850	POP	PSW	;Recover user's device number in C
0318 C9	01851	RET		
	01852	;		
	02000	;#		
	02001-		byte stream	
1	02002 02003	7 • This w	outine outputs i	nitialization bytes to port
	02003			am has the following format:
	02004	, numbers	. me oyle stre	am nas the reacting remain
1	02005	;	DB PPH	Port number
	02007	;	DB nn	Number of bytes to output
	02008	;		H Bytes to be output
	02009	;	:	
	02010	;	: Repeat	ed
	02011	;	1	
1	02012	;	DB OOH	Port number of O terminates
	02013			
	02014		parameters	
	02015 02016		HL -> Byte str	e 2 m
	02018	;	ne -> byte str	5 311
	02017	; Output\$Byte\$Sti	ream:	
1	02019	OBS\$Loop:		
0319 7E	02020	MOV	A, M	;Get port number
031A B7	02021	ORA	A	;Check if OOH (terminator)
031B C8	02022	RZ		;Exit if at end of stream
031C 322503	02023	STA	OBS\$Port	;Store in port number below
031F 23	02024	INX	н	;HL -> count of bytes
0320 4E	02025	MOV	С,М	;Get count
0321 23	02026	INX	Н	;HL -> first initialization byte
	02027	;		
	02028	OBS\$Next\$Byte:	A M	Gat payt byta
0322 7E	02029 02030	MOV INX	А, М Н	;Get next byte ;HL -> next data byte (or port number)
0323 23	02030	1111	11	The Price data byte to port number?

Figure 8-10. (Continued)

/

0004 80	02031 02032		DB	OUT				
032 4 D3	02032	0BS\$Port	00	001				
0325 00	02034		DB	0		;<- Set	up in instruction above	
0326 OD	02035		DCR	С			down on byte counter	
0327 C22203	02036		JNZ		t\$Byte		next data byte	
032A C31903	02037 02038		JMP	OBS\$Loc	P	;Go back	k for next port number	
	02038	; ; #						
	02101	;	CONST -	Console	status			
	02102	;						
	02103	;					rced input pointer and	
	02104 02105	;					ropriate input buffer. te whether or not there	
	02105	;		waiting		to indica	te whether of hot there	
	02107	;						
	02108	;	Entry p	arameter	s: none.			
	02109	;						
	02110 02111	2	Exit pa	rameters	5			
	02112	;		A = 000)H if the	ere is no	data waiting	
	02113	;					ta waiting	
	02114	;					-	
	02115			========				
	02116	CONST:					;<=== BIOS entry point (standard)	
032D 2A5800	02117 02118	;			sole\$Inpu	ıt	;Get redirection word	
0330 116400	02118		LXI			le\$Addres		
0333 CD6F06	02120		CALL	Select	Device\$	「able	;Get device table address	
0336 C34708	02121		JMP	Get\$Inp	out\$Statu	15	;Get status from input device	
	02122 02200						; and return to caller	
	02200	; # ;						
	02202	;	CONIN -	- consol	le input			
	02203	7						
	02204	7					haracter for the console input	
	02205 02206	;					stances, this can be a character or from a previously stored	
	02206	;					ced" into the input streamsfor	
	02208	;					tem initialization routines.	
	02209	;					m any previously stored character	
	02210	7	string	in memor	y. It is	s used to	inject the current time and date	
	02211 02212	;	or a st	ning ass On even	sociated	with a fi	unction key into the console ring of "SUBMIT STARTUP" is	
	02213	,	forced	into the	e console	e input s	tream to provide a mechanism.	
	02214	;						
	02215	;					from whichever physical device	
	02216 02217	2		ration b		nsole inp	ut redirection word (see the	
	02219	:	Coningo	Inaction L	JIOCK).			
0339 00	02219	CONIN\$D	elay\$Ela	apsed:	DB	0	;Flag used during function key	
	02220						; processing to indicate that	
	02221						; a predetermined delay has	
	02222	-					; elapsed	
	02223 02224	;	=======					
	02225	CONIN:					;<=== BIOS entry point (standard)	
	02226	;======					.	
033A 2A8DOF	02227		LHLD		ed\$Inpu	t	;Get the forced input pointer	
033D 7E	02228		MOV ORA	A, M A			;Get the next character of input ;Check if a null	
033E B7 033F CA4703	02229 02230		JZ	A CONIN\$1	N⊝\$FI		;Uneck if a null ;Yes, no forced input	
0342 23	02231		INX	H			;Yes, update the pointer	
0343 228D0F	02232		SHLD	CB\$Ford	ed\$Inpu	t	; and store it back	
03 4 6 C9	02233		RET					
	02234 02235	; CONIN\$N	~#E1				;No forced input	
0347 2A5800	02235	CONTINEN	LHLD	CB\$Cons	sole\$Inp	цt	;Get redirection word	
034A 116400	02237		LXI	D,CB\$De	≥vice\$Tal	ole\$Addre	sses	
034D CD6F06	02238		CALL		Device\$;Get device table address	
0350 CD9106	02239		CALL	Get\$Ing	out\$Chara	acter	;Get next character from input devic	e
	02240					• Euroti	on key processing	
				Eurotic	on\$Key\$Le		;Check if first character of functio	'n
0353 FE18	02241		CPI					
0353 FE1B	02241 02242 02243		CPI	i diletti			; key sequence (normally escape)	
0353 FE1B 0355 C0 0356 F5	02242		CPI RNZ PUSH	PSW	2			

0357					
	211D00	02246	LXI	H,DT\$Function\$Delay	;Get delay time constant for
		02247			; delay while waiting for subsequent
		02248			; characters of function key sequence
		02249			: to arrive
035A	19	02250	ĎAD	D	,
035B		02251	MOV	Č, M	;Get delay value
	0600	02252	MVI	B,0	;Make into word value
				A.	;Indicate timer not yet out of time
035E		02253	XRA	A CONIN\$Delay\$Elapsed	findicate time, not yet out of time
035F	323903	02254	STA		
0362	217803	02255	LXI		ed ;Address to resume at after delay
0365	CD6D08	02256	CALL	Set\$Watchdog	;Sets up delay based on real time
		02257			; clock such that control will be
		02258			; transferred to specified address
		02259			; after time interval has elapsed
		02260	CONIN\$Wait\$for	\$Delay:	;Wait here until delay has elapsed
0368	3A3903	02261	LDA	CONIN\$Delay\$Elapsed	;Check flag set by watchdog routine
036B	B7	02262	ORA	Α	
	CA6803	02263	JZ	CONIN\$Wait\$for\$Delay	
		02264			
		02265	CONIN\$Check\$fd	r\$Function:	
034E	211900	02266	LXI	H, DT\$Character\$Count	Now check if the remaining characters
0001	211/00	02267	EAT		; of the sequence have been input
0270	10	02267	DAD	P	, of the sequence have been input
0372			MOV	ы А.М	;Get count of characters in buffer
0373		02269			
	FE02	02270	CPI	Function\$Key\$Length - 1	· Particolo all'anterio del la la della della della
0376	B28103	02271	JNC	CONIN\$Check\$Function	;Enough characters in buffer for
		02272		_	; possible function key sequence
0379	F1	02273	POP	PSW	;Insufficient characters in buffer
		02274			to be a function key, so return
		02275			; to caller with lead character
037A	C9	02276	RET		
		02277			
		02278	;		
		02279	; The fo	llowing routine is called	by the watchdog routine
		02280		he specified delay has el	
		02281	•	the spectrice dera, has er	
		02282	, CONIN\$Set\$Dela	v¢Elanced.	
0070	3EFF	02283	MVI	A, OFFH	;Indicate watchdog timer out of time
	323903	02284	STA	CONIN\$Delay\$Elapsed	, indicate watchdog timer out of time
0370	323703	02284	RET	CONTRADELAYACIAPSED	Return to watchdog routine
0380	6.9	02285			Return to watchdog rodtine
		02286	:		
		02288	CONIN\$Check\$Fu		;Save the current "get pointer"
	211700	02289	LXI	H,DT\$Get\$Offset	
0384		02290	DAD	D	; in the buffer
0385		02291	MOV	Α,Μ	;Get the pointer
0386	F5	02292	PUSH	PSW	;Save pointer on the stack
		02293			
0387	211700	02294	LXI	H,DT\$Get\$Offset	;Check the second (and possibly third)
	CDF007	02295	CALL	Get\$Address\$in\$Buffer	; character in the sequence
038D	46	02296	MOV		
				B,M	;Get the second character
		02297	104	в,м	;Get the second character
		02297			
	C5	02298	IF	Three\$Character\$Functio	'n
038E		02298 02299	IF PUSH	Three\$Character\$Functio B	n ;Save for later use
038E 038F	211700	02298 02299 02300	IF PUSH LXI	Three\$Character\$Functio B H,DT\$Get\$Offset	'n
038E 038F 0392	211700 CDF007	02298 02299 02300 02301	IF PUSH LXI CALL	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer	n ;Save for later use ;Retrieve the third character
038E 038F 0392 0395	211700 CDF007 C1	02298 02299 02300 02301 02302	IF PUSH LXI CALL POP	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B	n ;Save for later use ;Retrieve the third character ;Recover second character
038E 038F 0392	211700 CDF007 C1	02298 02299 02300 02301 02302 02303	IF PUSH LXI CALL POP MOV	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer	n ;Save for later use ;Retrieve the third character
038E 038F 0392 0395	211700 CDF007 C1	02298 02299 02300 02301 02302 02303 02304	IF PUSH LXI CALL POP	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B	n ;Save for later use ;Retrieve the third character ;Recover second character
038E 038F 0392 0395 0396	211700 CDF007 C1 4E	02298 02299 02300 02301 02302 02303 02304 02305	IF PUSH LXI CALL POP MOV ENDIF	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M	n ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3
038E 038F 0392 0395 0396 0397	211700 CDF007 C1 4E D5	02298 02299 02300 02301 02302 02303 02304 02305 02306	IF PUSH LXI CALL POP MOV ENDIF PUSH	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D	n ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer
038E 038F 0392 0395 0396 0397	211700 CDF007 C1 4E	02298 02299 02300 02301 02302 02303 02304 02305 02306 02307	IF PUSH LXI CALL POP MOV ENDIF	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D	n ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer = - CB≸Function\$Key≸Entry≸Size
038E 038F 0392 0395 0396 0397	211700 CDF007 C1 4E D5	02298 02299 02300 02301 02302 02303 02304 02305 02306	IF PUSH LXI CALL POP MOV ENDIF PUSH	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D	n ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer = - CB\$Function\$Key\$Entry\$Size ;Get pointer to function key table
038E 038F 0392 0395 0396 0397	211700 CDF007 C1 4E D5	02298 02299 02300 02301 02302 02303 02304 02305 02306 02307	IF PUSH LXI CALL POP MOV ENDIF PUSH	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table	<pre>Save for later use ;Save for later use ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer c CB\$Function\$Key\$Entry\$Size ;Get pointer to function key table ; in configuration block</pre>
038E 038F 0392 0395 0396 0396 0397 0398	211700 CDF007 C1 4E D5	02298 02299 02300 02301 02302 02303 02304 02305 02306 02306 02307 02308	IF PUSH LXI CALL POP MOV ENDIF PUSH	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table	n ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer = - CB\$Function\$Key\$Entry\$Size ;Get pointer to function key table
038E 038F 0392 0395 0396 0396 0397 0398	211700 CDF007 C1 4E D5 21B000	02298 02299 02300 02301 02302 02303 02304 02305 02306 02307 02308 02309 02310	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Entry	<pre>Particle Particle Particl</pre>
038E 038F 0392 0395 0396 0397 0398 0398	211700 CDF007 C1 4E D5 21B000	02298 02299 02300 02301 02302 02303 02304 02305 02306 02307 02308 02309 02310	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$Fu:	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Entry	<pre>Particle Particle Particl</pre>
038E 038F 0392 0395 0396 0397 0398 0398 0398	211700 CDF007 C1 4E D5 21B000 1111300	02298 02299 02300 02301 02302 02303 02304 02305 02306 02307 02308 02309 02310 02311 02312	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$Fu DAD	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Entry nction: D	<pre>save for later use ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer = - CB%Function%Key%Entry%Size ;Get pointer to function key table ; in configuration block %Size ;Get entry size ready for loop ;Move to next (or first) entry</pre>
038E 038F 0392 0395 0396 0397 0398 0398 0398	211700 CDF007 C1 4E 21B000 1111300 19 7E	02298 02299 02300 02301 02302 02303 02304 02305 02306 02307 02308 02309 02310 02311 02312 02313	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$Fu: DAD MOV	Three\$Character\$Functio B H,DT\$Get\$Dffset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Entry nction: D A,M	<pre>n ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer : - CB\$Function\$Key\$Entry\$Size ;Get pointer to function key table ; in configuration block (\$Size ;Get entry size ready for loop ;Move to next (or first) entry ;Get second character of sequence</pre>
038E 0392 0395 0395 0396 0397 0398 0398 0398 0398 0398	211700 CDF007 C1 4E 21B000 111300 19 7E B7	02298 02299 02300 02301 02302 02303 02304 02305 02306 02306 02307 02308 02309 02310 02311 02312 02313 02314	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$Fu: DAD MOV ORA	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Entry nction: D A,M A	<pre> Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer - CB\$Function\$Key\$Entry\$Size ;Get pointer to function key table ; in configuration block \$Size ;Get entry size ready for loop ;Move to next (or first) entry ;Get second character of sequence ; Check if end of function key table </pre>
038E 038F 0395 0395 0396 0397 0398 0398 0398 0398 0398 0398 0396 0396 0396	211700 CDF007 C1 4E D5 21B000 1111300 19 7E B7 CAC203	02298 02299 02300 02301 02302 02303 02304 02305 02305 02306 02307 02308 02309 02310 02311 02312 02313 02314 02315	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$Fu MOV ORA JZ	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Entry nction: D A,M A CONIN\$Not\$Function	<pre>save for later use ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer e - CB%Function%Key%Entry%Size ;Get pointer to function key table ; in configuration block %Size ;Get entry size ready for loop ;Move to next (or first) entry ;Get second character of sequence ;Check if end of function key table ;Yes it is not a function key</pre>
038E 0392 0395 0395 0396 0397 0398 0398 0398 0398 0398 0398 0396 0340 0341 0344	211700 CDF007 C1 4E 21B000 111300 19 7E 87 CAC203 88	02298 02299 02300 02301 02302 02303 02304 02305 02306 02306 02307 02308 02309 02310 02311 02311 02313 02314	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Nex1\$Fu DAD MOV ORA JZ CMP	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Entry nction: D A,M A CONIN\$Not\$Function B	<pre>n ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer ; - CB\$FUnction\$Key\$Entry\$Size ;Get pointer to function key table ; in configuration block (\$Size ;Get entry size ready for loop ;Move to next (or first) entry ;Get second character of sequence ;Check if end of function key table ; Yes it is not a function key ;Compare second characters</pre>
038E 0392 0395 0395 0396 0397 0398 0398 0398 0398 0398 0398 0396 0340 0341 0344	211700 CDF007 C1 4E D5 21B000 1111300 19 7E B7 CAC203	02298 02299 02300 02301 02302 02303 02304 02305 02306 02306 02307 02308 02309 02310 02311 02312 02313 02314 02315 02316 02317	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$Fu MOV ORA JZ	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Entry nction: D A,M A CONIN\$Not\$Function	<pre>save for later use ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer e - CB%Function%Key%Entry%Size ;Get pointer to function key table ; in configuration block %Size ;Get entry size ready for loop ;Move to next (or first) entry ;Get second character of sequence ;Check if end of function key table ;Yes it is not a function key</pre>
038E 0392 0395 0395 0396 0397 0398 0398 0398 0398 0398 0398 0396 0340 0341 0344	211700 CDF007 C1 4E 21B000 111300 19 7E 87 CAC203 88	02298 02299 02300 02301 02302 02303 02304 02305 02306 02307 02308 02307 02308 02309 02310 02311 02312 02313 02314 02315 02316	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$FUI MOV ORA JZ CMP JNZ	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Table A,M A CONIN\$Not\$Function B CONIN\$Not\$Function	<pre>save for later use ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer e - CB%Function%Key%Entry%Size ;Get pointer to function key table ; in configuration block %Size ;Get entry size ready for loop ;Move to next (or first) entry ;Get second character of sequence ;Check if end of function key table ;Yes it is not a function key ;Compare second characters ;No match, so try next entry in table</pre>
038E 0392 0395 0395 0396 0397 0398 0398 0398 0398 0397 0399 0397 0340 0341 0344	211700 CDF007 C1 4E D5 21B000 111300 19 7E B7 CAC203 B8 5 C29E03	02298 02299 02300 02301 02302 02303 02304 02305 02305 02306 02307 02308 02309 02310 02311 02312 02313 02314 02315 02316 02318 02318	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$Fu: DAD MOV ORA JZ CMP JZ CMP	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Table D A,M A CONIN\$Not\$Function B CONIN\$Not\$Function B CONIN\$Next\$Function Three\$Character\$Function	<pre>n ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer e - CB\$FUnction\$Key\$Entry\$Size ;Get pointer to function key table ; in configuration block (\$Size ;Get entry size ready for loop ;Move to next (or first) entry ;Get second character of sequence ;Check if end of function key table ;Yes it is not a function key ;Compare second characters ;No match, so try next entry in table on</pre>
038E 0392 0395 0395 0396 0397 0398 0398 0398 0398 0398 0398 0394 0340 0341 0344	211700 CDF007 C1 4E D5 21B000 1111300 19 7E B7 CAC203 B8 CAC203 B8 C29E03	02298 02299 02300 02301 02302 02303 02303 02305 02305 02305 02307 02308 02307 02308 02310 02311 02312 02313 02314 02315 02314 02315 02318 02319 02320	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$Fu: DAD MOV ORA JZ JZ CMP JNZ IF INX	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Table CONIN\$Not\$Function B CONIN\$Not\$Function B CONIN\$Not\$Function H	<pre>save for later use ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer = - CB%Function%Key%Entry%Size ;Get pointer to function key table ; in configuration block *Size ;Get entry size ready for loop ;Move to next (or first) entry ;Get second character of sequence ;Check if end of function key table ;Yes it is not a function key ;Compare second characters ;No match, so try next entry in table on ;HL -> third character</pre>
038E 0392 0395 0395 0396 0397 0398 0398 0398 0398 0397 0399 0397 0340 0341 0344	211700 CDF007 C1 4E D5 21B000 111300 19 7E B7 CAC203 B8 CAC203 B8 C29E03	02298 02299 02300 02301 02302 02303 02304 02305 02305 02306 02307 02308 02309 02310 02311 02312 02313 02314 02315 02316 02318 02318	IF PUSH LXI CALL POP MOV ENDIF PUSH LXI CONIN\$Next\$Fu: DAD MOV ORA JZ CMP JZ CMP	Three\$Character\$Functio B H,DT\$Get\$Offset Get\$Address\$in\$Buffer B C,M D H,CB\$Function\$Key\$Table D,CB\$Function\$Key\$Table D A,M A CONIN\$Not\$Function B CONIN\$Not\$Function B CONIN\$Next\$Function Three\$Character\$Function	<pre>n ;Save for later use ;Retrieve the third character ;Recover second character ;Now BC = Char 2, Char 3 ;Save device table pointer e - CB\$FUnction\$Key\$Entry\$Size ;Get pointer to function key table ; in configuration block (\$Size ;Get entry size ready for loop ;Move to next (or first) entry ;Get second character of sequence ;Check if end of function key table ;Yes it is not a function key ;Compare second characters ;No match, so try next entry in table on</pre>

Figure 8-10. (Continued)

OBAB OP C Compare third characters OBAC C2F003 02325 UNX H HNMEXtFruction No match so try next entry in table OBAC C2F003 02325 UNX H iNL internet entry in table OBAC C2F003 02326 ENDIF internet entry in table internet entry in table OBAC C2F004 02330 02330 ENDIF internet entry in table OBAC C2F004 02330 02330 internet entry in table internet entry in table OBAC C2F004 02330 02330 internet entry in table internet entry in table OBAC C2F004 02330 02330 internet entry in table internet entry in table OBAC C2F004 02330 02330 internet entry in table internet entry in table OBAC C2F004 02330 02330 internet entry in table internet entry in table OBAC C2F004 02330 02330 internet entry in table internet entry in table OBAC C2F004 02330 02330 intery intable intable							
03AC 22803 02324 UNZ CONINNext#Function iNo match, so try next entry in table is the match found, compressue for is extra decrement 03B0 02325 INX H if the match found, compressue for is extra decrement 03B0 02329 ENDIF if the match found, compressue for is extra decrement 03B0 02330 INX H if the compressue for is extra decrement 03B1 02300 02331 SHLD CDBForced®Input if the compressue for is extra decrement 03B1 02335 INX H if the compressue for is extra decrement 02333 02335 Intermediate for is extra decrement if the stack must be is balaneed prior to return 03B5 02335 POP D If the stack must be is balaneed prior to return 03B5 02336 POP PSW If the compressue for its return 03B5 02340 POP PSW If the compressue for its return 03B5 02345 MOV A, M If the count If the count 03B5 02346 UNP MA If the count is extra decrement is used if the count is its returat its returat is its return	034P	B9	02323	CMP	с —		:Compare third characters
03AF 23 02325 INX H inher match found, compensate for 02327 03B0 23 02320 ENDIF istring of characters (00-byte ter 1 string of characters (00-byte ter 02321 03B1 228D0F 02332 INX H int> first characters (00-byte ter 02332 03B1 228D0F 02333 INX H istring of characters (00-byte ter 1 string of characters (00-byte ter 02332 03B4 D1 02335 NAU CB#Forced#Input istring of characters (00-byte ter 1 substitute string into the input istream 03B5 D1 02335 POP PSH Down the device table pointer 03B6 F1 02340 POP PSH Down the fact of set value 03B6 F2 02343 DAD D to reflect the character scount 03B7 F2 02344 MOV A.M ibound the count acter has already 03B7 F2 02344 MOV A.M ibe count acter has already 03B7 F2 02343 MOV A.M ibe cover device table pointer 03B7 F2 02344 MOV A.M ibe cover device table pointer 03B7 F2 02345 MOV A.M ibe cover device						\$Eunction	
02326 02327 ENDIF 02380 02328 ENDIF 02381 02382 INX H 0381 02382 INX H 0381 02383 SHLD CB#Forced*Input 1 The of character's (00-byte title operation of the construction of the construction of the input is ubtailing into the input operation of the stace must be operating into the input operation of the stace must be operating into the input is ublanced prior to return operating into the input operating into the input is ublanced prior to return operating is ublanced prior to re						an anecion	
02827 ENDIF 0380 023 02229 INX H :HL -> first characters (Orbita termination of the rest (Orbita termination characters termination characters (Orbita termination characters termination characters termination characters termination characters (Orbita termination characters termination characters termination characters termination characters (Orbita termination characters termination characters count is to reflect the characters count 02341 0386 FI 02337 DDD DD to reflect the characters count is to reflect the characters removed is thread the count offer 0387 C33003 02342 LXI H_DT\$Character\$Count iDowndate the character count is thread character has already is thread character is presumed to be part of 02351 0386 C33003 02384 UPP DOIN if the function keyseles is the function keyseles is thread thread termination is thread thread thre	USAF	23		1NX	П		
03B0 23 02329 INX H iHL -> first character of substitute 03B1 228D0F 02330 SHLD CB#Forced#Input istring of characters (00-byte ter 02331 SHLD CB#Forced#Input interms of characters (00-byte ter 02333 02334 istream interms of characters (00-byte ter 02334 istream interms of characters (00-byte ter 02335 02336 istream istream 02341 02337 POP D istream 0385 D1 02337 POP PSW ibum the "get" offset value 0385 02340 DAD D ibum the "get" offset value ibum the 'get" offset value 0387 21900 02341 LXI H,DT#Character#Count ibowdate the character is removed 0387 02345 DAD D ibowdate the character is removed ibowdate 0388 TE 02345 DAD D ibowdate ibowdate ibowdate 0387 21900 02346 DAD Feasorater ibowdate ibowdate ibowdate 0386 <							; extra decrement
03B0 23 02329 INX H ;HL -> first character of substitute 03B1 228D0F 02331 SHLD CB#Forced#Input ;False the COMIN routine inject the 02333 02333 ;SNLD CD#Forced#Input ;False the CoMIN routine inject the 02334 02335 ;SNLD CD#Forced#Input ;False ;SNLT 02335 02337 FOP D ;Celt the device table pointer ;SNLT 02341 02337 FOP PD ;Celt the device table pointer ;SNLT			02327	ENDIF			
0381 23800F0381 23800F0381 23800F0381 23800F0381 23800F0382330384 D102333023349000000000000000000000000000000000000			02328				
03B1 228DF 03B1 228DF 03B1 228DF 03B1 228DF 03B1 228DF 03B	03B0	23	02329	INX	н		;HL -> first character of substitute
03B1 228D0F 02331 SHLD CB#Forced#Input rHake the CONUN routine inject the processing into the input is stream 02333 02334 ream ream 02334 ream ream ream 02335 ream ream ream 02340 ream ream ream 0235 FI 02337 ream ream 0384 D1 02337 ream ream ream 0385 FI 02337 ream ream ream ream 0385 FI 02337 ream <							<pre>string of characters (00-byte term.)</pre>
 c) substitute string into the input c) substitute string into input c) substitute string input c) substitute string input c) substi	1960	229D0E		รมเก	CB\$Eorced\$	Input	Make the CONIN routine inject the
<pre> constructions and the stack must be construction sequence has be constructions and the stack must be con</pre>	0001	22000		SHED	0.041 01 0004	-input	
02334 02336 ;Now that a function sequence has be i identified, the stack must be propending of the stack must be							
02335 INOW that a function sequence has be 02337 0233 POP 0384 DI 02338 POP D 0385 F1 02339 POP PSU ;Dume the "qet" offset value 0385 F1 02339 POP PSU ;Dume the "qet" offset value 0386 F1 02343 DAD it or reflect the character count 0387 CI 02344 DAD it or reflect the character semovec 0388 F2 02344 DAD it or reflect the character semovec 0385 F2 02345 MOV A,M ;Get the count 0386 F2 02345 MOV A,M ;Get the count 0387 C2 02344 UIF Function%teysLength -1 ; the function key sequence 0387 C30A03 02348 UIF CONIN ; Return to CONIN processing to get 0387 C2 02350 CONIN*Not%Function: ; Attempts to recognize a function key sequence 02351 CONIN*Not%Function: ; Attempts to recognize a function key sequence 02352 CONIN*Not%Function: ; Herstraited, the set weil offset 02353 CONIN*Not%Function:							; stream
 identified, the stack must be identified,							
0384 DI02335 0386 FI02335 02339POP POP PSWDump the "get" offset value 0386 FI0385 FI02339 02341 02341 02341 02341 02342LXI DD DD;Dump the "get" offset value ;Dump the "get" offset pointer ;Deen deducted)0386 FC 30A03 0387 C3A03 0385 F702346 Q2349 Q2349 Q2350 Q2350 Q2350 Q2350 CONIN%Not%Function: ;Attempts to recognize a function key sequen ; have failed. The "get" offset pointer ; her function sequence are not lost. (2355 ; the function sequence are not lost. (2356 ; the function sequence are not lost.03C2 DI 03C3 F1 03C3 F1 03C3 F1 03C3 F1 03C4 211700 03C5Q255 Q256 Q256POP P D p FSW ; Recover previous "get" offset ; Recover previous "get" offset ; Recover lead character ; Recover lead character to the us Q2650 ; This routine outputs data characters to the console, device(s). Q2661 ; Console output Q2672 ; Console output Q2673 ; DOP Q2673 ; DOP Q2674 ; It also "traps" escape sequences being output to the console, Q2674 ; It also "traps" escape sequences to the console, device(s). Q2673 ; Dire the character is output and the device is flagged as being supended. Q2673 ; Console output and the device is flagged as being supende							
0385 F1 02339 POP PSW ;Dump the "get" offset value 0386 F1 02339 POP PSW ;Dump the "get" offset value 0237 211900 02342 LXI H,DT*Character*Count ;Dump the "get" offset value 02384 19 02343 DAD ; to reflect the character count 02384 19 02343 DAD ; to reflect the character semove ; from the buffer 02386 25 02344 MOV A,M ;DetChion*Ker%Length -1 ; been ducted) 0386 C502 02347 MOV A,M ;DetChion*Ker%Length -1 ; been ducted) 0386 C502 02348 UMP CONIN ;DetChion*Ker%Length -1 ; been ducted) 0387 C7 02347 MOV A,M ;DetChion*Ker%Length -1 ; been ducted) 0388 C33A03 02349 UMP CONIN ; the forced input character's 02350 CONIN*Not%Function: 02351 ;Attempts to recognize a function key sequer 02352 ; have failed. The "get" offset pointer mus 02353 02354 ; the character(s) presumed to be part of 02355 ; the function sequence are not lost. 02356 DOP PSW ;Recover device table pointer 02357 POP D ;Recover device table pointer 02358 F1 02357 POP D ;Recover device table pointer 02362 D1 02357 POP PSW ;Recover previous "get" offset 02362 PD 02360 DAD D * ; the function sequence are not lost. 02362 PD 02360 POP PSW ;Recover lead character 02362 S7 02361 MOV M,A ;Reset "get" offset in table 02363 ; This routine outputs data characters to the console device(s). 02364 ; it also "traps" scape sequences being output to the console; 02365 ; 02360 ; This routine outputs data characters to the console device(s). 02504 ; it also "traps" is cape sequences being output to the console; 02505 ; This routine outputs data character to all of the 02506 ; It naddition to output the set character to all of the 02507 ; devices currently selected in the console device(s). 02508 ; In addition to output the selected divice has not been 02510 ; it checks to see that output to the selected divice has not been 02511 ; suspended by XON/XOFF protocol, and that DTR is high if 02512 ; it should be. 02513 ; Once the character to be output 02521 ; CONOUT*Character; DE 0 ;Save area for character to be output 02520 ; 02520 ; CONOUT			02336				
 0385 F1 0235 0236 F1 0234 1 restored to its previous value so that 0235 1 restored to its previous value so that 0235 1 restored to its previous value so that 0235 1 restored to its previous value so that 0235 1 restored to its previous value so that 0235 1 restored to its previous value so that 0235 1 restored to its previous value so that 1 restoredi			02337				; balanced prior to return
 O385 F1 O2339 POP PSW Dump the "get" offset value O386 F1 O2340 O2341 O2341 O2341 O2342 LXI H, DT*Character*Count Downdate the character count to reflect the character removed from the bunction Sequence the count <li< td=""><td>03B4</td><td>D1</td><td>02338</td><td>POP</td><td>D</td><td></td><td>;Get the device table pointer</td></li<>	03B4	D1	02338	POP	D		;Get the device table pointer
0386 F1 0234 POP PSW ; Dume the function sequence lead cha 0387 121900 02342 LXI H.DT*Character*Count ; Downdate the character count 02384 19 02343 DAD ; to reflect the characters removed ; from the buffer 02385 72 02345 MOV A.M ; Get the count 0386 72 02346 SUI Function*Key%Length -1 ; (the lead character has already 0387 72 02347 MOV M.A 02386 77 02347 MOV M.A 02350 CONNENDOt%Function: 02350 CONNENDOt%Function: 02350 CONNENDOt%Function: 02351 ; the forced input characters 02350 CONNENDOt%Function: 02352 ; the character(s) presumed to be part of 02356 ; the function sequence are not lost. 02356 POP PSW ; Recover device table pointer 02356 POP PSW ; Recover device table pointer 02356 DAD D ; Recover device table pointer 02356 POP PSW ; Recover lead character to table 0368 77 02361 MOV M.A 0376 10 22359 LXI H.DT*Get#Offset 0376 10 22350 DAD D ; the lead character was detected 0376 19 02360 DAD D ; the lead character was detected 0376 19 02363 POP PSW ; Recover lead character was detected 0376 10 22350 DAD D ; the lead character was detected 0376 10 22350 DAD D ; the lead character was detected 0376 10 2236 DAD D ; the lead character was detected 0376 10 2236 DAD D ; the lead character was detected 0376 10 2236 DAD D ; Recover lead character was detected 0376 10 2236 DAD D ; the lead character was detected 0376 0 2236 DAD D ; the lead character was detected 0376 0 2236 ; # 02500 ; # 02500 ; # 02500 ; # 02501 ; Console output 02502 ; 02503 ; This routine outputs data characters to the console, 02505 ; triggering specific actions according to the sequences. 02506 ; A primitive "state-machine" is used to ster through escape 02507 ; sequence recognition. 02516 ; there the aracter has been output, if ETX/ACK protocol is in use, 02516 ; thardter is output and the device is flagged as being suspended. 02517 ; ' Entry parameters 02518 ; CONOUT*Character; DE 0 ;Save area for character to be output 02521 ; CONOUT*Character; DE 0 ;Save area for character to be output 02522 ; 02	03B5	F1	02339	POP	PSW		;Dump the "get" offset value
 0387 211900 02342 LXI H,DT\$Character\$Count ;Downdate the character count ; to reflect the characters removed ; from the buffer ; to reflect the characters removed ; from the buffer ; dot reflect the character has already 0380 D&O 02344 0380 D&O 02345 MOV A,M ; is the count ; the lead character has already 0380 D&O 02347 02348 0380 D&O 02349 02349 02349 02349 02340 02349 02340 02340 02340 02341 02341 02342 02342 02342 02343 02344 02344 02345 02346 02347 02348 02348 02349 02349 02349 02349 02340 02349 02340 02340 02341 02341 02341 02342 1 the function Key Sequence in processing to get in the recognize a function hey sequence in the inset of the recognize a function hey sequence in the inset of the in				POP			
0387 211900 02342 LXI H,DT\$Character\$Count ;Downdate the character count 0234 19 02343 DAD D ; to reflect the character semoved ; from the buffer 0386 72 02345 MOV A, H ;Get the count 0386 77 02347 MOV M, A ; been deducted 0386 77 02347 MOV M, A ; been deducted 0386 77 02347 MOV M, A ; the forced input characters 02349 ; the forced input character 02350 CONINNNOt%Function: 02350 CONINNNOt%Function: 02350 CONINNNOt%Function: 02350 ; the forced input character 02353 ; testored to its previous value so that 02350 CONINNNOt%Function: 02350 ; the forced input character 02350 ; the function sequence are not lost. 02350 ; the character(s) presumed to be part of 02350 ; the character so that 02350 POP PSH ; the character so its previous value so that 02350 POP PSH ; the cover previous "get" offset to the us after 02362 POP PSH ; the lead character was detected 0369 F1 02360 DAD D ; the lead character was detected 0369 F1 02363 POP PSW ; the lead character to the us 02365 ; triggering specific actions according to the sequences. 02366 ; A primitive "state-machine" is used to step through escape 02507 ; sequence recognition. 02506 ; A triggering specific actions according to the sequences. 02506 ; A triggering specific actions according to the sequences. 02507 ; sequence recognition. 02508 ; In addition be, 02501 ; Character is output and the device is flagged as being suspended by X0NXXOFF protocol, and that DTR is high if 02511 ; is should be. 02511 ; character is output and the device is flagged as being suspended. 02511 ; character is DE 0 ;Save area for character to be output 02520 ; 02521 ; CONNUT%Character; DE 0 ;Save area for character to be output 02520 ; 02523 ; DONNUT%Character; DE 0 ;Save area for character to be output 02520 ; 02524							,
 038A 19 02343 04D 0 c) reflect the characters removed ; from the buffer 038E D602 02346 038E T7 02347 038E T7 02348 038F C33A03 02349 02349 02340 02340 02341 02342 02342 02344 02344 02345 02345 02346 02347 02347 02348 02348 02349 02349 02340 02340 02344 02344 02344 02344 02345 02345 02345 02350 CONINNOt#Function: c) control (1000) <li c)<="" td=""><td>0007</td><td>2112000</td><td></td><td>1 7 7</td><td>U DT#Chause</td><td>a have the carries</td><td>. Decembric the absyrator court</td>	0007	2112000		1 7 7	U DT#Chause	a have the carries	. Decembric the absyrator court
02344; from the buffer0380 7E02345MOVA, H; Get the count0380 7702347MOVA, H; been deducted)0385 7702348JMPCONIN; Return to CONIN processing to get0235002349; the forced input characters0235102352; Attempts to recognize a function key sequer02352: Attempts to recognize a function key sequer02353; the character(s) presumed to be part of02354; the character(s) presumed to be part of02355; the character(s) presumed to be part of02354; the character(s) presumed to be part of02355; the character(s) presumed to be part of023621002357POPpresumed to be part of023621002357POPpresumed to be part of0236310; Recover device table pointer0236411presumed to be02375POPpresumed to be036271023590371902360DAD0377102360037702361MOV0377100377100378700379100370110370120371902363037012037190236303701203719023630370130371902363037013037112 <t< td=""><td></td><td></td><td></td><td></td><td></td><td>icter#count</td><td>; Downdate the character count</td></t<>						icter#count	; Downdate the character count
0388 7E 02345 MOV A,M ; Get the count 0380 D600 02344 SUIT Function%KeyLength -1; (the lead character has already 038F C33A03 02349 MOV MA ; The forced input character; 0385 D600 02346 JMP CONIN ; The forced input character; 02350 CONIN%tofFunction: ; The forced input character; ; The character(s) presumed to be part of 02351 ; The character(s) presumed to be part of ; The character(s) presumed to be part of 02354 ; The character(s) presumed to be part of 02354 ; The function sequence are not lost. 02355 ; The function sequence are not lost. 02354 ; The cover device table pointer 0362 F1 02357 POP 0363 F1 02357 POP 0370 19 02360 DAD ; HL -> "get" offset in table 0371 19 02361 MOV M.A ; Recover lead character was detected 0370 19 02363 POP FSW ; Recover lead character to the us 0376 19 02363 POP PSW ; Recover lead character to the us<	03BA	14		DAD	D		
 03BC D602 02346 SUI Function\$Key\$Length -1 ; (the lead character has already 03BF 77 02347 MOV M,A ; been deducted) 02350 0201N ; Beturn to CONIN processing to get 02350 0201N ; Return to CONIN processing to get 02351 :Attempts to recognize a function key sequer 02352 : have failed. The "get" offset pointer mus 02354 : the character(s) presumed to be part of 02355 : the character(s) presumed to be part of 02356 : the character(s) presumed to be part of 02357 : the character(s) presumed to be part of 02358 : the character(s) presumed to be part of 02357 : the character(s) presumed to be part of 02350 : the character(s) presumed to be part of 02357 : the character vas detected 0336 : 02360 : DAD D : presumed to be part of 1 : Pope D : Recover previous "get" offset to it has 0336 : 02361 : 02362 : 02363 : 02363 : 02364 : 02365 : 02366 : 02366 : 02367 : 02360 : 02368 : 02369 <li: 00019<="" li=""> : 02369 <li: 00019<="" li=""> <li: 000019<="" li=""> : 02360 <li: 00001<="" td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td></li:></li:></li:></li:>							
03BE 7702347MOVM,A: been deducted)03BF C33A0302348UMPCONIN;Return to CONIN processing to get0235002349if the forced input characters0235102350CONIN\$Not\$Function:;Atempts to recognize a function key sequer02352if the character(s) presumed to be part of02355; the character(s) presumed to be part of0235302356; the character(s) presumed to be part of0235502354; the character(s) presumed to be part of0235502355; the function sequence are not lost.0235602357POPp0363 F102358POPPSW0367 F102360DADpHL -> "get" offset in table0367 9702361MOVM,A; field character was detected0367 9702361MOVM,A; field character was detected0362 7702361MOVM,A; field character was detected0362 7102363POPPSW; field character0364RET; field character was detected;0365 ;; This routine outputs data characters to the console device(s).;02503 ;This routine outputs data characters to the sequences.02504 ;; triagering specific actions according to the sequences.02505 ;if riggering specific actions according to the sequences.02505 ;if riggering specific actions according to the sequences.02505 ;if riggering specific actions according to the sequences. <tr< td=""><td></td><td></td><td>02345</td><td></td><td>Α,Μ</td><td></td><td></td></tr<>			02345		Α,Μ		
03BE 7702347MOVM,A: been deducted)03BF C33A0302348UMPCONIN;Return to CONIN processing to get0235002349if the forced input characters0235102350CONIN\$Not\$Function:;Atempts to recognize a function key sequer02352if the character(s) presumed to be part of02355; the character(s) presumed to be part of0235302356; the character(s) presumed to be part of0235502354; the character(s) presumed to be part of0235502355; the function sequence are not lost.0235602357POPp0363 F102358POPPSW0367 F102360DADpHL -> "get" offset in table0367 9702361MOVM,A; field character was detected0367 9702361MOVM,A; field character was detected0362 7702361MOVM,A; field character was detected0362 7102363POPPSW; field character0364RET; field character was detected;0365 ;; This routine outputs data characters to the console device(s).;02503 ;This routine outputs data characters to the sequences.02504 ;; triagering specific actions according to the sequences.02505 ;if riggering specific actions according to the sequences.02505 ;if riggering specific actions according to the sequences.02505 ;if riggering specific actions according to the sequences. <tr< td=""><td>03BC</td><td>D602</td><td>02346</td><td>SUI</td><td>Function\$K</td><td>(ey\$Length −1</td><td>; (the lead character has already</td></tr<>	03BC	D602	02346	SUI	Function\$K	(ey\$Length −1	; (the lead character has already
O3BF C33A03O2348JMPCONIN; Return to CONIN processing to get ; the forced input characters02350CONINNOt\$Function:; the forced input characters02351; Attempts to recognize a function key sequer ; have failed. The "get" offset pointer mus ; the character(s) presumed to be part of ; the character was detected 02650 ; the character was detected 02650 ; the presumed to the console device(s). ; the character was detected 02501 ; Console output 02502 ; 02503 ; This routine outputs data characters to the console, 02505 ; triggering specific actions according to the sequences. 02506 ; A primitive "state-machine" is used to step through escape 02507 ; sequence recognition. 02508 ; In addition to output ing the console output redirection word, 02510 ; it character has been output, if ETX/ACK protocol is in							
02349 ; the forced input characters 02351 :Attempts to recognize a function key sequer 02352 ; have failed. The "get" offset pointer musical 02353 ; the character(s) presumed to be part of 02354 ; the function sequence are not lost. 02355 ; the function sequence are not lost. 02356 ; the function sequence are not lost. 03357 POP presover device table pointer 03357 02356 ; the function sequence are not lost. 03357 POP presover device table pointer 03357 02357 POP 03357 02350 LXI 03357 02360 DAD 03357 02361 MOV 03357 02362 pret" offset as it was after 0357 02362 pret" offset as it was after 0359 1 02363 POP 0350 1 Console output ; Reset "get" offset as it was after 0350 1 Console output ; Reset reget offset 0350 1 Console output ; Reset reget offset 0350 1<							
02350CONTINENOLSFUNCtion:02351; Attempts to recognize a function key sequences02352; have failed. The "get" offset pointer mus02353; the character(s) presumed to be part of02354; the character(s) presumed to be part of02355; the character(s) presumed to be part of02356; the character(s) presumed to be part of02357POPpercenter0358POP; the character(s) presumed to be part of0359Q2357POPpercenter035102358POPpercenter0352Q2359LXIH,DT\$Get\$Offset0357Q2360DADpercenter03587Q2361MOVM,A03587Q2361MOVM,A0359POPPSW; Recover lead character was detected03507Q2364RET; Return the lead character was detected03501Console output; Recover lead character was detected03531it also "traps" escape sequences being output to the console,02501;Console output02503;Ti algering specific actions according to the sequences.02504;It also "traps" escape sequences being output to the console,02505;ti algering specific actions according to the sequences.02505;ti algering specific actions according to the sequences.02507;sequence recognition.02508;In addition to output ing the console outp	JODP	000400		Of the	0.0114.11		
02351;Attempts to recognize a function key sequence02352; have failed. The "get" offset pointer mus02353; the character(s) presumed to be part of02354; the function sequence are not lost.02355; the function sequence are not lost.02356; the function sequence are not lost.02357POP; Recover device table pointer0362pression of the function sequence are not lost.036202357POP0363F102358037102360DAD03687702361037102362pression of fist as it was after0369f1023630364RET; Return the lead character was detected0365;;0364; Console output0370; Mathematic actions according to the sequences.0364; In also "traps" escape sequences being output to the console,0370; Mathematic actions according to the sequences.0370; Mathematic actions according to the sequences.0370; In also "traps" escape sequences being output to the console,02503; This routine outputs data characters to the console device(s).02504; In addition to outputting the next character to all of the02505; triggering specific actions according to the secape02506; devices currently selected in the console output redirection word,02507; sequence recognition.02508; In addition to outputting the next character to all of the02509				CONTRANSION			, the forced input characters
 bit failed. The "get" offset pointer must of a sequence are not lost. color of the sequence are is a sequence are sequences being output to the console. color of the sequence are not lost. color of the sequence				CONTINANOLAFUNCI	ion:		- A - A
02353 ; restored to its previous value so that 02354 ; the character(s) presumed to be part of 02355 ; the function sequence are not lost. 0354 ; the function sequence are not lost. 0355 ; the function sequence are not lost. 0356 ; the function sequence are not lost. 0357 POP ; Recover device table pointer 0356 POP ; Recover previous "get" offset 0357 POP ; Recover previous "get" offset 0357 POP ; Recover previous "get" offset 0357 POP ; Recover lead character 0357 POP ; Recover lead character 0358 POP ; Recover lead character 0358 POP ; Recover lead character 0359 ; Recover lead character safter 0359 ; Recover lead character safter 0359 ; Recover lead character safter 0350 ; This routine outputs data characters to the console device(s). ; 02501 ; Console output ; supering specific actions according to the sequences. 02502 ; it also						;Attem	pts to recognize a function key sequence
02354 ; the character(s) presumed to be part of 02355 ; the function sequence are not lost. 0302 D1 02357 POP D ;Recover device table pointer 0303 F1 02358 POP PSW ;Recover previous "get" offset 0303 F1 02359 LX1 H,DT\$Get\$Offset ;HL -> "get" offset in table 0307 F1 02360 DAD D ;HL -> "get" offset as it was after 0308 F1 02363 POP PSW ;Recover lead character was detected 0305 F1 02363 POP PSW ;Recover lead character was detected 0305 F1 02363 POP PSW ;Recover lead character to the us 0306 F1 02363 FOP PSW ;Recover lead character to the us 02302 ; 02501 ; Console output ;Recover lead character to the us 02502 ; 1 talso "traps" escape sequences being output to the console, 02505 ; triggering specific actions according to the sequences. 02503 ; 1 talso "traps" escape sequences being output to the console, 02507 ; sequence recognition. 02505 ; 1 talso "traps" escape sequences being output to the console, 02507 ; sequence recognition.							
 c) c) c			02353			; res	tored to its previous value so that
03C2 D1 02357 POP D ;Recover device table pointer 03C3 F1 02358 POP PSW ;Recover previous "get" offset 03C4 211700 02359 LXI H,DT\$Get\$Offset ;HL -> "get" offset in table 03C7 19 02360 DAD D ;HL -> "get" offset in table 03C8 77 02361 MOV M,A ;Reset "get" offset in table 03C8 77 02363 POP PSW ;Recover lead character was detected 03C9 F1 02363 POP PSW ;Recover lead character 03C4 C9 02364 ; Console output ;Recover lead character 02500 ;# Console output ;Recover lead character ;Recover lead character 02501 ; Console output ; ;Recover lead character ;Recover lead character 02503 ; This routine outputs data characters to the console device(s). ; ;Recover lead character 02503 ; it also "traps" escape sequences being output to the console, ; ; 02503 ; it also "traps" escape is equences being output to the console, ; ; 02505 ;			02354			; the	character(s) presumed to be part of
03C2 D1 02357 POP D ;Recover device table pointer 03C3 D1 02358 POP PSW ;Recover previous "get" offset 03C4 211700 02359 LXI H,DT%Get%Offset ;HL -> "get" offset in table 03C7 19 02360 DAD D ;HL -> "get" offset in table 03C8 77 02361 MOV M,A ;Reset "get" offset in table 03C9 F1 02363 POP PSW ;Recover lead character was detected 03C9 F1 02363 POP PSW ;Recover lead character 03C4 C9 02364 ; Console output ;Recover lead character 02500 ;# Console output ;Recover lead character ;Recover lead character 02501 ; Console output ; ;Recover lead character ;Recover lead character 02503 ; This routine outputs data characters to the console device(s). ; ;Recover lead character 02503 ; it also "traps" escape sequences being output to the console, ; ; 02503 ; it also "traps" escape is equences being output to the console, ; ; 02505 ;			02355			: the	function sequence are not lost.
 03C2 Di 02357 POP D ;Recover device table pointer 03C3 Fi 02358 POP PSW ;Recover previous "get" offset 03C4 211700 02359 LXI H,DT\$Get\$Offset 03C7 19 02360 DAD D ;HL → "get" offset in table 03C8 77 02361 MOV M,A ;Reset "get" offset as it was after 023C8 77 02363 POP PSW ;Recover lead character was detected 03C9 F1 02363 POP PSW ;Recover lead character was detected 03C6 79 02364 RET ;Recover lead character to the us 02500 ;# 02500 ;# 02503 ; This routine outputs data characters to the console device(s). 02504 ; It also "traps" escape sequences being output to the console, 02505 ; triggering specific actions according to the sequences. 02506 ; devices currently selected in the console output redirection word, 02509 ; devices currently selected in the console is no device has not been 02511 ; suspended by X0N/XOFF protocol, and that DTR is high if 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02515 ; character is output and the device is flagged as being suspended. 02516 ; 02517 ; 'Entry parameters 02518 ; 02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 CONOUT\$Character: DB 0 ;Save area for character to be output 02524 03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal 						,	
03C3 F1 02358 POP PSW ;Recover previous "get" offset 03C4 211700 02359 LXI H,DT%Get%Offset	0202	D 1		POP	n		*Recover device table pointer
03C4 211700 02359 LXI H,DT\$Get\$0ffset 03C7 19 02360 DAD D ;HL -> "get" offset in table 03C8 77 02361 MOV M,A ;Reset "get" offset in table 03C8 77 02361 MOV M,A ;Reset "get" offset in table 03C8 77 02361 MOV M,A ;Reset "get" offset in table 03C8 77 02361 MOV M,A ;Reset "get" offset in table 03C8 77 02363 POP PSW ;Return the lead character 03C4 C9 02364 RET ;Return the lead character to the us 02500 ;# 02500 ; 02501 02503 This routine outputs data characters to the console device(s). 02505 02504 It also "traps" escape sequences being output to the console, 02507 02505 triggering specific actions according to the sequences. 02507 02507 sequence recognition. 02508 In addition to outputting the next character to all of the 02510 it suspended by X0N/XOFF protocol, and that DTR is high if 02512 it is should be. 02513 Once the cha							Pacover providus "get" officet
O3C7 19O2360DADD;HL -> "get" offset in tableO3C8 77O2361MOVM,A;Reset "get" offset as it was afterO2362O2363POPPSW;Recover lead character was detectedO3C8 77O2364RET;Return the lead character to the usO3C6 C9O2364RET;Return the lead character to the usO2362 ;O2363 ;This routine outputs data characters to the console device(s).O2503 ;This routine outputs data characters to the console device(s).O2504 ;It also "traps" escape sequences being output to the console,O2505 ;triggering specific actions according to the sequences.O2506 ;A primitive "state-machine" is used to step through escapeO2507 ;sequence recognition.O2508 ;In addition to outputting the next character to all of theO2509 ;devices currently selected in the console output redirection word,O2510 ;it should be.O2513 ;Once the character has been output, if ETX/ACK protocol is in use,O2514 ;and the specified length of message has been output, an EtxO2515 ;character is output and the device is flagged as being suspended.O2518 ;O2510 ;O2520 ;O2520 ;O2518 ;CO2518 ;claracter to be outputO2520 ;O2512 ;O2521 ;CONOUT storage variablesO2522 ;O3CB 00O2523 CONOUT\$Character:O3CB 00O2525 CONOUT\$Processor:DW CONOUT\$Normal						***	, Necover previous get offset
03CS 7702361MOVM,A;Reset "get" offset as it was after02362; the lead character was detected03C9 F102363POPPSW;Recover lead character03C4 C902364RET;Return the lead character to the us02501 ;02365 ;;02502 ;02503 ;This routine outputs data characters to the console device(s).02503 ;02503 ;This routine outputs data characters to the console device(s).02504 ;It also "traps" escape sequences being output to the console,02505 ;triggering specific actions according to the sequences.02506 ;A primitive "state-machine" is used to step through escape02507 ;sequence recognition.02508 ;In addition to outputting the next character to all of the02510 ;it checks to see that output to the selected device has not been02511 ;suspended by XON/XOFF protocol, and that DTR is high if02512 ;it should be.02513 ;Once the character has been output, if ETX/ACK protocol is in use,02514 ;and the specified length of message has been output, an Etx02517 ;'Entry parameters02518 ;C = character to be output02520 ;02521 ;03C8 000253303C8 000253303C8 000253403C8 000252503C0 DB030252503C0 DB030252503C0 DB0302525						irrset	
02362; the lead character was detected03C9 F102363POPPSW;Recover lead character03CA C902364RET;Return the lead character to the us02350;#02500;#02501:Console output02502;02503:This routine outputs data characters to the console device(s).02504:It also "traps" escape sequences being output to the console,02505:triggering specific actions according to the sequences.02506:A primitive "state-machine" is used to step through escape02507:sequence recognition.02508:In addition to outputting the next character to all of the02510:it checks to see that output to the selected device has not been02511:suspended by XON/XOFF protocol, and that DTR is high if02512:it should be.02513:Once the character has been output, if ETX/ACK protocol is in use,02514:and the specified length of message has been output, an Etx02518:.02520:.02521:Convourt storage variables02522:.030002523CONOUT storage variables03252:.03254:.03255CONOUT storage variables03254:.03255:.03254:.03254:.032					-		;HL -> "get" offset in table
03C9 F1 02363 POP PSW ;Recover lead character 03CA C9 02364 RET ;Return the lead character to the us 02365 ; 02500 ;# 02501 ; Console output 02502 ; 02503 ; This routine outputs data characters to the console device(s). 02503 ; It also "traps" escape sequences being output to the console, 02505 02505 ; triggering specific actions according to the sequences. 02506 ; A primitive "state-machine" is used to step through escape 02507 ; sequence recognition. 02508 ; In addition to outputt to the selected device has not been 02510 ; it checks to see that output to the selected device has not been 02510 ; it checks to see that output, if ETX/ACK protocol is in use, 02511 ; suspended by X0N/XOFF protocol, and that DTR is high if 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02517 ; 'Entry parameters 02518 ; C 02519 ;	0308	77	02361	MOV	M,A		
03C9 F1 02363 POP PSW ;Recover lead character 03CA C9 02364 RET ;Return the lead character to the us 02365 ; 02500 ;# 02501 ; Console output 02502 ; 02503 ; This routine outputs data characters to the console device(s). 02503 ; It also "traps" escape sequences being output to the console, 02505 02505 ; triggering specific actions according to the sequences. 02506 ; A primitive "state-machine" is used to step through escape 02507 ; sequence recognition. 02508 ; In addition to outputt to the selected device has not been 02510 ; it checks to see that output to the selected device has not been 02510 ; it checks to see that output, if ETX/ACK protocol is in use, 02511 ; suspended by X0N/XOFF protocol, and that DTR is high if 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02517 ; 'Entry parameters 02518 ; C 02519 ;			02362				; the lead character was detected
03CA C9 02364 RET ;Return the lead character to the us 02300 ;# 02501 ; Console output 02502 ; 02503 ; This routine outputs data characters to the console device(s). 02503 ; It also "traps" escape sequences being output to the console, 02505 ; triggering specific actions according to the sequences. 02506 ; A primitive "state-machine" is used to step through escape 02507 ; sequence recognition. 02508 ; In addition to outputting the next character to all of the 02509 ; devices currently selected in the console output redirection word, 02512 ; it should be. 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02517 ; 'Entry parameters 02528 ; 02519 C = character to be output 02520 ; 02517 ; 'Entry parameters 02518 ; 02520 ; 02521 ; CONOUT storage variables	0309	F1	02363	POP	PSW		
02365 ; 02500 ;# 02501 ; Console output 02502 ; 02503 ; This routime outputs data characters to the console device(s). 02504 ; It also "traps" escape sequences being output to the console, 02505 ; triggering specific actions according to the sequences. 02506 ; A primitive "state-machine" is used to step through escape 02507 ; sequence recognition. 02508 ; In addition to outputting the next character to all of the 02509 ; devices currently selected in the console output redirection word, 02510 ; it checks to see that output to the selected device has not been 02511 ; suspended by X0N/XOFF protocol, and that DTR is high if 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output. an Etx 02515 ; character is output and the device is flagged as being suspended. 02517 ; ' Entry parameters 02518 ; 02519 ; C = character to be output 02520 ; 02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 CONOUT\$Character: DB 0 ;Save area for character to be output 02524 03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal				RET			
02500 ;# 02501 ; 02502 ; 02503 ; 02503 ; 02503 ; 02503 ; 02503 ; 02503 ; 02504 ; 02505 ; 02506 ; 02507 ; 02508 ; 02509 ; 02509 ; 02509 ; 02500 ; 11 addition to outputing the next character to all of the 02509 ; devices currently selected in the console output redirection word, 02510 ; it checks to see that output to the selected device has not been 02511 ; suspended by XON/XOFF protocol, and that DTR is high if 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02517 ; Entry parameters 02518 ;							,
02501;Console output02502;02503;This routime outputs data characters to the console device(s).02503;It also "traps" escape sequences being output to the console,02505;It also "traps" escape sequences being output to the console,02506;A primitive "state-machine" is used to step through escape02507;sequence recognition.02508;In addition to outputting the next character to all of the02509;devices currently selected in the console output redirection word,02510;it checks to see that output to the selected device has not been02512;it should be.02513;Once the character has been output, if ETX/ACK protocol is in use,02514;and the specified length of message has been output, an Etx02515;character is output and the device is flagged as being suspended.02519;C = character to be output02521;CONOUT storage variables02522;003CE DB0302525CONOUT\$Processor:03CC DB0302525CONOUT\$Processor:				• #			
02502;02503;This routine outputs data characters to the console device(s).02504;It also "traps" escape sequences being output to the console,02505;triggering specific actions according to the sequences.02506;A primitive "state-machine" is used to step through escape02507;sequence recognition.02508;In addition to outputting the next character to all of the02509;devices currently selected in the console output redirection word,02510;it checks to see that output to the selected device has not been02512;it should be.02513;Once the character has been output, if ETX/ACK protocol is in use,02514;and the specified length of message has been output, an Etx02517;'Entry parameters02518;C = character to be output02520;0252102521;CONOUT storage variables02522;003CB 0002523CONOUT\$character:03CC DB0302525CONOUT\$Processor:04C DB0302525CONOUT\$Processor:04C DB0302525CONOUT\$Processor:04C DB0302525CONOUT\$Processor:04C DB0302525CONOUT\$Processor:04C DB0302525CONOUT\$Processor:04C DB0302525							
02503 ; This routime outputs data characters to the console device(s). 02504 ; It also "traps" escape sequences being output to the console, 02505 ; triggering specific actions according to the sequences. 02506 ; A primitive "state-machine" is used to step through escape 02507 ; sequence recognition. 02508 ; In addition to outputting the next character to all of the 02509 ; devices currently selected in the console output redirection word, 02510 ; it checks to see that output to the selected device has not been 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02517 ; / Entry parameters 025218 ; C = character to be output 025219 ; CONOUT storage variables 02522 ; ;					σαιραι		
02504 it also "traps" escape sequences being output to the console, 02505 triggering specific actions according to the sequences. 02506 A primitive "state-machine" is used to step through escape 02507 sequence recognition. 02508 In addition to outputting the next character to all of the 02509 devices currently selected in the console output redirection word, 02510 it checks to see that output to the selected device has not been 02512 it should be. 02513 Once the character has been output, if ETX/ACK protocol is in use, 02514 and the specified length of message has been output, an Etx 02515 character is output and the device is flagged as being suspended. 02517 ' Entry parameters 02519 C = character to be output 02520 CONOUT storage variables 02521 CONOUT storage variables 02522 03CB 00 02525 03CC DB03 02525 CONOUT\$Processor:							
02505 : triggering specific actions according to the sequences. 02506 : A primitive "state-machine" is used to step through escape 02507 : sequence recognition. 02508 : In addition to outputting the next character to all of the 02509 : devices currently selected in the console output redirection word, 02510 : it checks to see that output to the selected device has not been 02511 : suspended by XON/XOFF protocol, and that DTR is high if 02512 : it should be. 02513 : Once the character has been output, if ETX/ACK protocol is in use, 02514 : and the specified length of message has been output, an Etx 02515 : character is output and the device is flagged as being suspended. 02517 : Entry parameters 02518 : C 02519 : C = character to be output 02520 : 02521 02521 : CONOUT storage variables 02522 : . 03CB 00 02524 . 03CC DB03 02525							
02506 : A primitive "state-machine" is used to step through escape 02507 ; sequence recognition. 02508 : In addition to outputting the next character to all of the 02509 ; devices currently selected in the console output redirection word, 02510 ; it checks to see that output to the selected device has not been 02511 ; suspended by XON/XOFF protocol, and that DTR is high if 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02515 ; character is output and the device is flagged as being suspended. 02516 ; 02517 02519 ; C = character to be output 02520 ; 02520 02521 ; CONOUT storage variables 02522 ; 0 03CE DB03 02525 CONOUT\$Processor: 03CC DB03 02525 CONOUT\$Processor:							
02507 ; sequence recognition. 02508 ; In addition to outputting the next character to all of the 02509 ; devices currently selected in the console output redirection word, 02510 ; it checks to see that output to the selected device has not been 02511 ; suspended by X0N/X0FF protocol, and that DTR is high if 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02515 ; character is output and the device is flagged as being suspended. 02517 ; Entry parameters 02519 ; C = character to be output 02520 ; 02521 02521 ; CONOUT storage variables 02522 ; 03CB 00 02524 03CE DB03 02525 CONOUT\$Processor: DW 03CC DB03 02525 CONOUT\$Processor: DW				; trigger	ing specifi	c actions ac	cording to the sequences.
02508 In addition to outputting the next character to all of the 02509 devices currently selected in the console output redirection word, 02510 : it checks to see that output to the selected device has not been 02511 : suspended by X0N/X0FF protocol, and that DTR is high if 02512 : it should be. 02513 : Once the character has been output, if ETX/ACK protocol is in use, 02515 : character is output and the device is flagged as being suspended. 02517 : 'Entry parameters 02518 : C = character to be output 02519 : C = character to be output 02520 : 0000UT storage variables 02521 : CONOUT storage variables 02522 : 0 03CE DB03 02525 CONOUT\$Processor: 03CC DB03 02525 CONOUT\$Processor:							used to step through escape
02508 In addition to outputting the next character to all of the 02509 devices currently selected in the console output redirection word, 02510 : it checks to see that output to the selected device has not been 02511 : suspended by X0N/X0FF protocol, and that DTR is high if 02512 : it should be. 02513 : Once the character has been output, if ETX/ACK protocol is in use, 02515 : character is output and the device is flagged as being suspended. 02517 : 'Entry parameters 02518 : C = character to be output 02519 : C = character to be output 02520 : 0000UT storage variables 02521 : CONOUT storage variables 02522 : 0 03CE DB03 02525 CONOUT\$Processor: 03CC DB03 02525 CONOUT\$Processor:			02507	; sequenc	e recogniti	on.	
02509 devices currently selected in the console output redirection word, 02510 it checks to see that output to the selected device has not been 02511 suspended by XON/XOFF protocol, and that DTR is high if 02512 it should be. 02513 Once the character has been output, if ETX/ACK protocol is in use, 02514 and the specified length of message has been output, an Etx 02515 character is output and the device is flagged as being suspended. 02516 02517 02519 C = character to be output 02520 02521 02521 CONOUT storage variables 02522 0 03CB 00 02523 0200000000000000000000000000000000000			02508	; In addi	tion to out	putting the	next character to all of the
02510 it checks to see that output to the selected device has not been 02511 suspended by XON/XOFF protocol, and that DTR is high if 02512 it should be. 02513 Once the character has been output, if ETX/ACK protocol is in use, 02514 and the specified length of message has been output, an Etx 02515 character is output and the device is flagged as being suspended. 02517 ' 02518 : 02519 C = character to be output 02520 : 02521 CONOUT storage variables 02522 : 03CB 00 02523 02524 ONOUT\$Character: 03CC DB03 02525 0200 CONOUT\$Processor: 03CC DB03 02525							
02511 ; suspended by XON/XOFF protocol, and that DTR is high if 02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02515 ; character is output and the device is flagged as being suspended. 02516 ; 02517 ; Entry parameters 02519 ; C = character to be output 02520 ; 02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 020UT\$Character: DB 0 03CC DB03 02525 0200UT\$Processor: DW CONOUT\$Normal							
02512 ; it should be. 02513 ; Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02515 ; character is output and the device is flagged as being suspended. 02516 ; . 02517 ; Entry parameters 02519 ; C = character to be output 02520 ; . 02521 ; CONOUT storage variables 02522 ; . 03CB 00 . . 02524 . . 03CC DB03 . . 02525 . . 03CC DB03 . . 02525 . . 03CC DB03 . . 02525 . . 02526 . . 02527 . . 02528 . . 02524 . . 02525 . . 02526 . <td< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td></td<>							
02513 : Once the character has been output, if ETX/ACK protocol is in use, 02514 ; and the specified length of message has been output, an Etx 02515 ; character is output and the device is flagged as being suspended. 02517 ; Entry parameters 02518 ; C = character to be output 02519 ; C = character to be output 02520 ; 02521 02521 ; CONOUT storage variables 02522 ; 0 03CB 00 02523 CONOUT\$character: DB 0 ;Save area for character to be output 03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal						on protocol	, and that but is high in
02514 ; and the specified length of message has been output, an Etx 02515 ; character is output and the device is flagged as being suspended. 02516 ; 02517 ; Entry parameters 02518 ; 02519 ; 02520 ; 02521 ; 02522 ; 02522 ; 03CB 00 02524 03CC DB03 02525 0200UT\$Processor: DW C0NOUT\$Normal							
02515 ; character is output and the device is flagged as being suspended. 02516 ; 02517 ; 02518 ; 02519 ; 02520 ; 02521 ; 02522 ; 02522 ; 03CB 00 02524 ; 03CB 02525 02000 ; 03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal							
02516 ; 02517 ; Entry parameters 02518 ; 02519 ; C = character to be output 02520 ; 02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 CONOUT\$Character; DB 0 ;Save area for character to be output 03CC DB03 02525 CONOUT\$Processor; DW CONOUT\$Normal							
02517 ; ' Entry parameters 02518 ; 02519 ; C = character to be output 02520 ; 02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 CONOUT\$Character: DB 0 ;Save area for character to be outpu 02524 02524 02524 02525 CONOUT\$Processor: DW CONOUT\$Normal				; charact	er is outpu	it and the de	vice is flagged as being suspended.
02517 ; ' Entry parameters 02518 ; 02519 ; C = character to be output 02520 ; 02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 CONOUT\$Character: DB 0 ;Save area for character to be outpu 02524 02524 02524 02525 CONOUT\$Processor: DW CONOUT\$Normal			02516	,			
02518 ; 02519 ; C = character to be output 02520 ; 02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 CONOUT\$Character: DB 0 ;Save area for character to be outpu 02524 03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal				; / Entryp	arameters		
02519 ; C = character to be output 02520 ; 02521 ; 02522 ; 03CB 00 02523 0200 CONOUT\$Character: DB 0 02524 03CB 00 02523 03CB 00 02524 03CC DB03 02525 000UT\$Processor: DW C0N0UT\$Normal					-		
02520 ; 02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 CONOUT\$Character: DB 0 ;Save area for character to be outpu 02524 03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal				•	C = charac	ter to be ou	tput
02521 ; CONOUT storage variables 02522 ; 03CB 00 02523 CONOUT\$Character: DB 0 ;Save area for character to be outpu 02524 03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal					s anaras		
02522 ; 03CB 00 02523 CONOUT\$Character: DB 0 ;Save area for character to be outpu 02524 03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal					storage ver	iables	
03CB 00 02523 CONOUT#Character: DB 0 ; Save area for character to be outpu 03CC 02524 0				, CONOUT	storage var	190162	
02524 03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal				7			a and the second se
03CC DB03 02525 CONOUT\$Processor: DW CONOUT\$Normal	03CB	00		CONCUT\$Characte	r: DB	; Q	;Save area for character to be output
ADED/ This is the adducer of the picco of	0300	DB03	02525	CONOUT\$Processo	er: DW	CONOUT	
vzozo ; mis is the address of the piece of			02526				;This is the address of the piece of
02527 ; code that will process the next							
02528 ; character. The default case is							
						· ·	
03CE 0000 02530 CONOUT\$String\$Pointer: DW 0 ;This points to a string (normally	03CE	0000		CONCUT\$String\$P	ointer: DW	0	FINIS POINTS TO A STRING (NORMALLY
02531 ; in the configuration block) that							
02532 ; is being preset by characters fro							; is being preset by characters from
02533 ; the console output stream							the console output stream

Figure 8-10. (Continued)

		00504			This and the second sec
03D0	00	02534 02535	CONOUT\$String	Kength: DB 0	;This contains the maximum number of ; characters to be preset into a
		02536			; from the console output stream
		02537			
		02538	;		
		02539		ARNING ***	
		02540 02541	; The o	utput error message routin utine. On entry here, the	data byte to be output
		02542		scine. On entry nere, the	DE registers set up correctly.
	-	02543		be on the stack, and the b	c registers set up convective
		02543	1		
		02545	, CONOUT\$0EM\$En	+++++	
0301	32CB03	02546		CONOUT\$Character	;Save data byte
	C3E803	02547	JMP	CONOUT\$Entry2	;HL already has special bit map
0001	002000	02548	;	0011001121111/2	, 1
		02549	;======================================		
		02550	CONOUT:		(IOS entry point (standard)
		02551	; =====================================		
0307	2ACC03	02552	LHLD	CONOUT\$Processor	;Get address of processor to handle
		02553			; the next character to be output
		02554	,		;(Default is CONOUT\$Normal)
03DA	E9	02555	PCHL		;Transfer control to the processor
		02556	ŧ		
		02557	;		
		02558	CONOUT\$Normal		;Normal processor for console output
03DB		02559	MOV	A,C	;Check if possible start of escape
	FE1B	02560	CPI	Function\$Key\$Lead	; sequence
03DE	CA1204	02561	JZ	CONOUT\$Escape\$Found	;Perhaps
		02562	CONOUT\$Forced		
03E1		02563	MOV	A,C	;Forced output entry point
03E2	32CB03	02564	STA	CONOUT\$Character	;Not escape sequence Save data byte
		02565			
03E5	2A5A00	02566	LHLD	CB\$Console\$Output	;Get console redirection word
		02567	;		
		02568	CONOUT\$Entry2	; ;<=== c	output error message entry point
0055		02569	; 		sses :Addresses of dev. tables
	116400	02570	LXI PUSH	D,CB\$Device\$Table\$Addre N	Put onto stack ready for loop
03EB		02571	PUSH	L H	;Put onto stack ready for 100p
03EC	63	02572 02573	PUSH	п	
		02573	CONQUT\$Next\$D	evice.	
03ED	E1	02575	POP	H H	Recover redirection bit map
03ED		02576	POP	n g	;Recover device table addresses pointer.
OBEE	CD6F06	02576	CALL	D Select\$Device\$Table	;Get device table in DE
03F2		02578	ORA	A	;Check if a device has been
0012		02579	SILL		; selected (i.e. bit map not all zero)
03E3	CAODO4	02580	JZ	CONOUT\$Exit	No, exit
03F6		02581	PUSH	B ;Yes - B	;Save redirection bit map
03F7		02582	PUSH	н	;Save device table addresses pointer
		02583	CONCUT\$Wait:		
03F8	CDOF06	02584	CALL	Check\$Output\$Ready	;Check if device not suspended and
		02585			; (if appropriate) DTR is high
03FB	CAF803	02586	JZ	CONOUT\$Wait	;No, wait
		02587			
03FE	F3	02588	DI		;Interrupts off to avoid
		02589			; involuntary re-entrance
	3ACB03	02590	LDA	CONOUT\$Character	Recover the data byte
0402		02591	MOV	C, A	Ready for output
	CD2608	02592	CALL	Output\$Data\$Byte	;Output the data byte
0406	FB	02593	EI		
		02594	_		
		02595	CALL	Process\$Etx\$Protocol	;Deal with Etx/Ack protocol
	CD3A06		JMP	CONQUT\$Next\$Device	;Loop back for next device
	CD3A06 C3ED03	02596			
		02597			
040A	C3ED03	02597 02598	CONOUT\$Exit:		- Deserves data at succession
040A 040D	C3ED03	02597 02598 02599	LDA	CONOUT\$Character	Recover data character
040A 040D 0410	C3ED03 3ACB03 79	02597 02598 02599 02600	LDA MOV	CONOUT\$Character A,C	;Recover data character ;CP/M "convention"
040A 040D	C3ED03 3ACB03 79	02597 02598 02599 02600 02601	LDA		
040A 040D 0410	C3ED03 3ACB03 79	02597 02598 02599 02600 02601 02602	LDA MOV RET	A, C	;CP/M "convention"
040A 040D 0410 0411	C3ED03 3ACB03 79 C9	02597 02598 02599 02600 02601 02602 02603	LDA MOV RET ; CONOUT\$Escape	A,C \$Found:	;CP/M "convention" ;Possible escape sequence
040A 040D 0410 0411	C3ED03 3ACB03 79	02597 02598 02599 02600 02601 02602 02603 02604	LDA MOV RET ; CONOUT\$Escape LXI	A,C \$Found: H,CONOUT\$Process\$Escape	;CP/M "convention"
040A 040B 0410 0411 0412	C3ED03 3ACB03 79 C9 211904	02597 02598 02599 02600 02601 02602 02603 02604 02605	LDA MOV RET ; CONOUT\$Escape LXI CONOUT\$Set\$Pr	A,C \$Found: H,CONOUT\$Process\$Escape ocessor:	;CP/M "convention" ;Possible escape sequence ;Vector processing of next character
040A 040D 0410 0411 0412 0415	C3ED03 3ACB03 79 C9 211904 22CC03	02597 02598 02599 02600 02601 02602 02603 02604 02605 02606	LDA MOV RET ; CONOUT\$Escape LXI CONOUT\$Set\$Pr SHLD	A,C \$Found: H,CONOUT\$Process\$Escape ocessor:	;CP/M "convention" ;Possible escape sequence ;Vector processing of next character ;Set vector address
040A 040B 0410 0411 0412	C3ED03 3ACB03 79 C9 211904 22CC03	02597 02598 02599 02600 02601 02602 02603 02603 02605 02605 02606 02607	LDA MOV RET ; CONOUT\$Escape LXI CONOUT\$Set\$Pr SHLD RET	A,C \$Found: H,CONOUT\$Process\$Escape ocessor:	;CP/M "convention" ;Possible escape sequence e ;Vector processing of next character
040A 040D 0410 0411 0412 0415	C3ED03 3ACB03 79 C9 211904 22CC03	02597 02598 02599 02600 02601 02602 02603 02604 02605 02606 02606 02607 02700	LDA MOV RET CONOUT\$Escape LXI CONOUT\$Set\$Pr SHLD RET ;#	A,C \$Found: H,CONOUT\$Process\$Escape ocessor:	;CP/M "convention" ;Possible escape sequence ;Vector processing of next character ;Set vector address
040A 040D 0410 0411 0412 0415	C3ED03 3ACB03 79 C9 211904 22CC03	02597 02598 02599 02600 02601 02602 02603 02603 02605 02605 02606 02607	LDA MOV RET ; CONOUT\$Escape LXI CONOUT\$Set\$Pr SHLD RET ;# ;	A,C \$Found: H,CONOUT\$Process\$Escape ocessor:	;CP/M "convention" ;Possible escape sequence ;Vector processing of next character ;Set vector address ;Return to BIOS caller

00703 00704 00706 00707 00700 00700 00700 00700 00700 00700 00700 00700 00700 00700 00700 00700 00700						
0019 02100 0019 02100 1000000000000000000000000000000000000		02703	•			
0419 21180 02705 0419 21180 02706 0410 57 i after escape in C i Get base of reconition table i Get base of reconition table i Get base of reconition table i Concurse i Get base i Concurse i Get base of reconition table i Concurse i Get base i Concurse i Get base i Concurse i Get base i Concurse i Get base i Get base i Get base i Get base			CONCUTERS	*		·Canturl suvives have with showsates
O419 211802 02706 LXI H_CONDUT%Escape#Table ; Det base of recognition table O410 TF 02707 MOV A.M ; Deck if at end of table O410 TF 02708 MOV A.M ; Deck if at end of table O416 CA2804 02711 CONUT%estelentry: ; Deck if at end of table O422 CA3804 02712 JZ CONUT%estelentry: ; Deck if at end of table O422 CA3804 02713 INX H ; Move to next entry in table O422 CA3804 02713 INX H ; Move to next entry in table O422 CA3804 02713 UNX H ; Move to next entry in table O422 CA18 02720 INX H ; Move to next entry in table O422 CA18 02727 UNUT%HowHatch: ; No match found, so original O422 CA18 02727 CALL CONUT%Forced ; Dutual to console devices O422 CA18 02727 CALL CONUT%Forced ; Dutual to console devices O422 CA18 02727 CALL CONUT%Forced ; Dutual to conormal <tr< td=""><td></td><td></td><td>CONDOTAFTOCESS</td><td>acscape:</td><td></td><td></td></tr<>			CONDOTAFTOCESS	acscape:		
OPECTOR CONDUTSWEETERTY: 0410 7E 02707 CONDUTSWEETERTY: 0410 7E 02709 ORA A 0410 7E 02709 ORA A 0410 7E 02709 ORA A 0420 52 02711 JZ CONDUTSMEETERTY 0422 62 02713 INX H 0426 22 02714 JZ CONDUTSMEETERTY IOb back and check again 0428 02 02717 JP CONDUTSMEETERTY IOb back and check again 0428 02 02717 CONDUTSMEETERTY IOb back and check again Smatt be output 0428 02 02710 PUSH B INE Smatt be output 0428 02 02721 PUSH B INE Smatt be output 0428 02 02722 CONUTSMEETERTY IOb console devices IOb console devices 0428 02 02723 CALL CONUTFForced IOb console devices IOb console devices 0432 02 02730 INF H. CONUTSMEETERTY IOb console devices<						
OAID PT OBJOR MOV A.M :Check if at end of table OAID DET CONDUT®NOTMICH :Yes. no match found Compare context character OAID DET CONDUT®NotMitch :Yes. no match found Compare context character OAID DET CONDUT®NotMitch :Yes. no match found Compare context character OAID DET CONDUT®NotMitch :Ther match How to next entry in table OAID DET CONDUT®Next#Entry :Do back and check again :Do context context entry in table OAID CONDUT®NotMatch: :No match found, so original :escape and following character OAID CONDUT®NotMatch: :No match found, so original :escape and following character OAID CONDUT®NotMatch: :Not be output :escape and following character OAID CONDUT®NotMatch: :escape and following character OAID CONDUT®NotMatch: :escape if output :escape OAID CONDUT®NotMatch: :escape :escape OAID CONDUT®Notmal :Set vector back to normal	0419 211802				\$Table	;Get base of recognition table
0418 07 02709 ORA A 0418 07 02709 U CONDUTSNOTMatch :Yes. no match found 0418 07 02711 U CONDUTSNotMatch :Pompare to data character 0420 07 02713 U CONDUTSNotMatch :Pompare to data character 0420 02 02713 UNX H Move to next entry in table 0422 02 02715 UNX H Move to next entry in table 0422 02 02715 UNX H Move to next entry in table 0422 02 02715 UNX H : escape and following character 0422 02103 02724 PUSH B : imatch found, so original 0422 02103 02725 CALL CONUTSet Normal : Dutnut to console devices 0433 01803 02725 CALL CONUTSet Normal : Set vector back to normal 0435 01803 02726 INX H : : : 0435 021803 02736 INX H <			CONOUT\$Next\$En	try:		
$\begin{array}{ccccc} 0421 & B^{\circ} \\ 0422 & C^{\circ} \\ 0421 & B^{\circ} \\ 0422 & C^{\circ} \\ 0421 & B^{\circ} \\ 0422 & C^{\circ} \\ 042 & C^{\circ$	041C 7E	02708		A, M		;Check if at end of table
Odd1 D9 C/HP C :Compare to data character Odd2 Odd2 Odd2 COMOUTMatch :They match Odd2 Odd2 Odd2 Odd2 Odd2 Odd2 Odd2 PUSH B :stage and following character Odd2 Odd2 Odd2 FUB :stage and following character Odd2 Odd2 Odd2 FUB :stage and following character Odd2 Odd2 Odd2 FUB :stage and following character Odd2 Odd2 FUB :stage and following character Odd2 Odd2 COMOUTSetetForced :stage	041D B7	02709	ORA	A		
Odd1 D9 C/HP C :Compare to data character Odd2 Odd2 Odd2 COMOUTMatch :They match Odd2 Odd2 Odd2 Odd2 Odd2 Odd2 Odd2 PUSH B :stage and following character Odd2 Odd2 Odd2 FUB :stage and following character Odd2 Odd2 Odd2 FUB :stage and following character Odd2 Odd2 Odd2 FUB :stage and following character Odd2 Odd2 FUB :stage and following character Odd2 Odd2 COMOUTSetetForced :stage	041E CA2B04	02710	JZ	CONDUT\$No\$Match		:Yes, no match found
0422 23 02712 JZ CONDUTSMatch iThey match 0425 23 02714 INX H iMove to next entry in table 0426 23 02714 INX H iMove to next entry in table 0427 23 02714 INX H iMove to next entry in table 0428 C51004 02715 OMUTSMext%Entry iSo back and check again 0428 C51004 02715 CONDUTSMext%Entry iNo match found, so original 0428 C5 02712 PUSH B iSave character after escape 0428 C5103 02724 PUSH C.Function%Forced iDutu to consolf evices 0428 C5103 02725 CALL CONOUTSForced iDutu to consolf evices 0428 C5104 02725 CONOUTSForced iDutu to consolf evices 0428 C5104 02725 CONOUTSForced iDutu to consolf evices 0428 C5104 02733 INX H :HL => LS byte of subprocessor 0428 C5104 02735 INX H :Dutu 0428 C51000078 PCHL CONOU				C C		
0425 23 02713 INK H iffwore to next entry in table 0426 23 02714 INK H iffwore to next entry in table 0426 23 02715 JNK H CONOUTSNextSEntry 160 back and check again 0428 C3100 02715 CONOUTSNextSEntry 160 back and check again ifficience 0428 C31 02720 CONOUTSNextSEntry 160 back and check again ifficience 0428 C31 02720 CONOUTSNextSEntry 150 back character ifficience 0428 C31 02721 PUSH B Fister character ifficience 0431 C11 02722 CONOUTSForced 100tputuit, too ifficience ifficience 0432 C31504 02725 CALL CONOUTSForced joutputit, too ifficience 0435 C11503 02723 LXI H_CONOUTSEctForcessor ifficience ifficience 0435 211503 02734 MOV E, H ifficience ifficience 0438 23 02735 INX H ifficience <t< td=""><td>0422 042804</td><td></td><td></td><td></td><td></td><td></td></t<>	0422 042804					
0422 2302714INXH0427 2302715INXH0428 C31C0402716JMPCONOUTSNEXtSEntryf06 back and check again0428 C31C0402716CONOUTSNEXtSEntryf06 back and check again0428 C502720FUSHB: escape and following character0428 C502721PUSHB: faste character after escape0422 0E10302722CALLCONOUTSForced: Dutput to console devices0430 CE10302723CALLCONOUTSForced: Dutput to console devices0432 C211902724CALLCONOUTSForced: Dutput to console devices0432 C2110302725CALLCONOUTSForced: Dutput to console devices0432 C2110402726LX1H, CONOUTSSetSForcessor: for console devices0432 C2110402726LX1H, CONOUTSSetSForcessor: for console devices0432 C2110402731CONOUTSAttProcessor: for console of subprocessor0432 C2110402732CONOUTSAttProcessor: for console input stream (using i force input stream (using i stream (using i force input stream (using i force input stream (using i stream (using i force input stream (using i force input stream (using i the console input stream (using i force input stream (using i force input stream (using i stream (using i force input stream (using i the console input stream (using i force input	0422 CASB04					
0427 23 02715 INX H 0428 C31C04 02715 JMP CONDUTSNESSITE ; No match found, so original 02717 ; The anticle in the interval i						;Move to next entry in table
0428 C31C04 02716 02717 02718 UMP CONDUT\$Next\$Entry ; Do back and check again 0428 C5 02718 02718 CONDUT\$Next\$Entry ; Do back and check again 0428 C5 02721 CONDUT\$Next\$Entry ; Do back and check again 0428 C5 02721 PUSH B ; Escape and following character 0420 0E18 02722 MVI C.Function%(reykled) ; Get escape character scape 0420 0E18 02723 CALL CONDUT\$Forced ; Dutput it, too 0431 C1 02724 POP B ; Get character scape ; Dutput it, too 0432 0E180 02727 CALL CONDUT\$Forced ; Dutput it, too 0435 210B03 02773 CONUT\$Set\$Forcesor ; for subsequent characters 0435 22 02730 LXI H, CNOUT\$Normal ; Set vector back to normal 0436 23 02737 CONUT\$Normal ; Set vector back to normal 0437 25 02737 CONUT\$Normal ; Set vector back to normal 0438 23 02737 XCHG ; HL -> Subprocessor 02740 02737						
02717 02718 ; No match found, so original ; escape and following character ; must be output ; Suev character after escape ; Output it choice is devices ; Output it choice is device is deviced it put ; Output it choice is device is deviced it put ; Output it choice is device is device ; output it choice is device is device ; output it choice is device ;						
02717 02718 ; No match found, so original ; escape and following character ; must be output ; must be output ; must be output ; must be output ; escape and following character ; output to console devices ; output to console input ; output to console input stream (using ; console input stream (using ; console input stream (using ; console input stream (using ; console input stream ; output is console input stream ; output stream; output stream ; output stream; output stream; output ; and storing the int be device; ; output stream; output stream; ; output stream; output stream; ; outpu	0428 C31C04	02716	JMP	CONOUT\$Next\$Ent	ry	;Go back and check again
02718 CONDUT\$No\$Match: ; No match found, so original 02720 02720 ; must be output 0428 C5 02721 PUSH B ; must be output 0426 C5 02722 MVI C.Function\$Key\$Lead ; Det escape character 0426 CDE103 02723 CALL CONDUT\$Forced ; Dutput to console devices 0427 CDE103 02725 CALL CONDUT\$Forced ; Dutput to console devices 0432 CDE103 02725 CALL CONDUT\$Forced ; Dutput to console devices 0435 218003 02727 CONDUT\$Forced ; Dutput to console devices 0435 21803 02727 UMP CONDUT\$Forced ; Dutput to console input 0435 21802 02727 UMP CONDUT\$Forced ; Dutput to console input 0438 C31504 02775 INN H ; HL -> LS byte of subprocessor 0438 C31504 02738 PCHL ; Dutpocessor ; Dutpocessor 0438 C31504 02738 PCHL ; Subprocessor ; Dutpocessor 0440 E9 02738 PCHL <td></td> <td>02717</td> <td>;</td> <td></td> <td></td> <td></td>		02717	;			
02219 ; escape and following character 0428 C5 02271 PUSH B 0420 0E18 02272 PUSH B ;save character after escape 0420 0E18 02723 CALL COMOUTSForced ;Dutput to console devices 0432 0E103 02725 CALL COMOUTSForced ;Dutput to console devices 0432 0E103 02725 CALL COMOUTSForced ;Dutput it, too 02726 ;CALL COMOUTSetsthormal ;Set vector back to normal 0438 02739 ;LXI H,COMOUTSetsthormal ;Set vector back to normal 0438 02730 ;OUNOUTSetsthormal ;Set vector back to normal 0438 02735 INN H ;HL -> LS byte of subprocessor 0438 02737 INN H ;Get HS byte 0438 02737 XCHG ;HL -> LS byte of subprocessor 0437 02737 XCHG ;HL -> Subprocessor 0438 02744 CONOUTSEtsthorcedStInput ;Get HS			CONCUT\$No\$Matel	.		:No match found, so original
042E C5 02720 push B j must be output 042E C5 02721 PUSH B j Save character after escape 042E C0EL03 02723 CALL CONDUT*Forced jOutput to console devices 0431 C1 02724 POP B jSet escape character after escape 0431 C1 02724 CONDUT*Forced jOutput is, too 0432 CDE103 02724 CONDUT*Forced jOutput is, too 0435 21DE03 02723 CONDUT*Set*Normal: jSet vector back to normal 0438 C31504 02723 CONOUT*Set*Processor ; for subsequent characters 02731 02733 INX H. CONOUT*Set*Processor ; for subsequent characters 0438 C5 02734 MOV H ; for subprocessor ; for subprocessor 0438 C5 02735 MOV H ; into console input stream (using ; forced input) 0440 E9 02735 CONOUT*Set*Forced*Input ; subprocessor to inject time into ; into console input stream 0441 218FOF 02745 CONOUT*Set*Forced*Input ; console input strea						
0428 C502721PUSHB;Save character after escape0422 0E10302723CALLCONOUT\$Forced;Output to console devices0431 C102724POPB;Get escape character after escape0432 0E10302725CALLCONOUT\$Forced;Output it, too02725CALLCONOUT\$Forced;Output it, too02726;CALLCONOUT\$Forced;Output it, too02727CONOUT\$Forced;Output it, too0273002729UNPCONOUT\$Set\$Processor; for subsequent character start0435 21DE0302723LXIH.CONOUT\$Set\$Processor; for subsequent character start0435 21DE0302724UNPCONOUT\$Set\$Processor; for subsequent character start0438 2302733INXH;HL -> LS byte of subprocessor0430 2302734MOVD.H; for det MS byte0435 2402735INXH; for subsequent character start0436 2502736MOVD.H; for det MS byte0436 2502736MOVD.H; for subsequent character start0437 2202736PCHL; for console input stream0438 21800F02743LXIH, Date02741:into console input stream0442 22800F02745SHLD02745::0444 22800F02745LXI02745::0444 22800F02745LXI02745:: <tr< td=""><td></td><td></td><td></td><td></td><td></td><td></td></tr<>						
0422 0212 MUL C.Function%Key%Lead jGet escape character 0432 0210 02723 CALL CONOUT#Forced jOutput to console devices 0431 0110 02724 FOP B jOutput to console devices 0432 0210 02725 CALL CONOUT#Forced jOutput to console devices 0432 02126 LXI H.CONOUT#Sormall ;Set vector back to normal 0438 02723 CONOUT#Set#Normall ;Set vector back to normal 0438 02730 ; CONOUT#Set#Processor ; for subsequent characters 0438 02735 LXI H.CONOUT#Set#Processor ; for subprocessor 0438 02735 INX H ; HL -> LS byte of subprocessor 0438 02737 NCHG ; HL -> LS byte of subprocessor 0437 02737 NCHG ; HL -> LS byte of subprocessor 0437 02737 NCHG ; JUP occessor to inject current date 0437 02737 CONOUT#Set#Forced#Input 0441 21		02720	DUCU	D		
042E CDE103 02723 CALL CONOUT#Forced ;Output to console devices 0431 C1 02724 FOP B ;Get character after escape 0432 CDE103 02725 CALL CONOUT#Forced ;Output it, too 0432 CDE103 02725 CALL CONOUT#Forced ;Output it, too 0435 210803 02727 LXI H, CONOUT#Set#Processor ; for subsequent character s 02731 02723 CONOUT#Set#Processor ; for subsequent character s 02731 02733 INX H ; HL -> LS byte of subprocessor 0438 23 02733 INX H ; Get LS byte 0430 55 02735 MV D, H ; Get MS byte 0438 24 02735 MV D, H ; Get MS byte 0430 52 02736 PCHL ; Subprocessor ; Goto subprocessor 0437 58 02747 CONOUT#Set#Forced#Input: ; Subprocessor to inject time into 02749 2274 CNOUT#Set#Forced#Input: ; Subprocessor to inject time into 02749				•		
0431 CI 02724 PDP B ; pot character after escape 0432 CDE103 02725 CALL CONUT#Set#Normal: ; putput it, too 0432 21DB03 02726 LXI H.CONUT#Set#Normal: ; set vector back to normal 0438 21DB03 02729 JMP CONUT#Set#Normal: ; set vector back to normal 0438 02730 0773 ; DMP CONUT#Set#Normal: ; set vector back to normal 0438 02733 0773 ; DMP CONUT#Set#Norcessor ; for subsequent character 0438 23 02733 INX H ; HL -> LS byte of subprocessor 0437 22736 MOV P.H ; Get MS byte ; subprocessor 0438 23 02737 XCHG ; HL -> LS byte of subprocessor 0438 56 02737 XCHG ; HL -> LS byte of subprocessor 0439 23 02735 INX H ; Bot console input stream (using ; into console input stream (using ; ito console input stream (usin					Lead	
0432 CDE103 02725 CALL CONOUT\$Forced ;Dutput it, too 0432 CDE103 02727 CONOUT\$Set\$Normal: Set vector back to normal 0438 C31504 02729 JHP CONOUT\$Set\$Freessor ; for subsequent characters 0438 C31504 02729 JHP CONOUT\$Set\$Freessor ; for subsequent characters 0438 C31504 02723 CONOUT\$Match: ; ; for subsequent characters 0437 21 CONOUT\$Match: ; for subsequent characters ; 0438 C3 02733 INX H ; HL -> LS byte of subprocessor 0437 C4 02734 MOV E,M ; Get LS byte 0438 C3 02735 INX H ; Get LS byte 0438 C4 02736 PCHL ; Souprocessor ; into console input stream (using 0440 E9 02742 ; Inte console input stream (using 0441 218FOF 02744 CONOUT\$Set\$Forced\$Input: ; Subprocessor to inject time into 02747 02744 CONOUT\$Set\$Forced\$Input: ; Subprocessor to inject time into 0444 21890F 02750 LXI H, Time\$In\$\$Set\$Forced\$Input ; Console input stream						
0435 21DB03 02726 LXI H.CONOUT\$Set\$Normal: ;Set vector back to normal 0438 02730 02730 ; for subsequent characters 0438 02730 ; 02731 ; 0438 02730 ; for subsequent characters 0438 02732 CONOUT\$Set\$Processor ; for subsequent characters 0438 02733 INX H ;HL -> LS byte of subprocessor 0438 02734 MOV E.M ;Get LS byte 0432 02735 INX H ;Get LS byte 0438 02736 MOV D.M ;Get MS byte 0437 EX MOV D.M ;Get MS byte 0438 ES 02735 ROU J.M ;Get MS byte 0437 ES 02738 PCHL ;Goto subprocessor 02740 CONOUT\$Set\$Forced\$Input ; <forced input<="" td=""> ; 0444 22800F 02745 SHLD CONOUT\$Set\$Porced\$Input ; <!--</td--><td>0431 C1</td><td>02724</td><td>POP</td><td>в</td><td></td><td>;Get character after escape</td></forced>	0431 C1	02724	POP	в		;Get character after escape
0435 21DB03 02726 LXI H.CONOUT\$Set\$Normal: ;Set vector back to normal 0438 02730 02730 ; for subsequent characters 0438 02730 ; 02731 ; 0438 02730 ; for subsequent characters 0438 02732 CONOUT\$Set\$Processor ; for subsequent characters 0438 02733 INX H ;HL -> LS byte of subprocessor 0438 02734 MOV E.M ;Get LS byte 0432 02735 INX H ;Get LS byte 0438 02736 MOV D.M ;Get MS byte 0437 EX MOV D.M ;Get MS byte 0438 ES 02735 ROU J.M ;Get MS byte 0437 ES 02738 PCHL ;Goto subprocessor 02740 CONOUT\$Set\$Forced\$Input ; <forced input<="" td=""> ; 0444 22800F 02745 SHLD CONOUT\$Set\$Porced\$Input ; <!--</td--><td>0432 CDE103</td><td>02725</td><td>CALL</td><td>CONOUT\$Forced</td><td></td><td>:Output it, too</td></forced>	0432 CDE103	02725	CALL	CONOUT\$Forced		:Output it, too
0435 21050 02727 CONOUT\$Set\$Normal: ;Set vector back to normal 0438 C31504 02729 JMP CONOUT\$Set\$Processor ; for subsequent characters 0438 C31504 02729 JMP CONOUT\$Set\$Processor ; for subsequent characters 0438 C31504 02730 ; . . . 0438 C31504 02733 INX H ;HL -> LS byte of subprocessor 0438 C31504 02735 INX H ;Get MS byte 0435 2105 02735 INX H ;Get MS byte 0436 C5 02737 XCHG ;HL -> Subprocessor . 0437 EB 02737 XCHG ;HL -> Subprocessor . 02740 CONOUT\$Date: ;Subprocessor to inject current date . 02741 ;Subprocessor to inject current date . . 02742 CONOUT\$Set\$Forced\$Input: . . . 0444 22800F 02745 SHLD CONOUT\$Set\$Forced\$Input . . 0444 22800F 02745 LXI			•			
0435 21DB03 02728 LXI H,CONDUT\$Normal ;Set vector back to normal 0438 C31504 02729 JHP CONDUT\$Set\$Processor ; for subsequent characters 0438 C31504 02732 CONDUT\$Natch: CONDUT\$Set\$Processor ; for subsequent characters 0438 Z3 02733 INX H :HL -> L5 byte of subprocessor 0438 Z3 02737 INX H :Get L5 byte 0438 Z3 02737 INX H :Get MS byte 0438 Z3 02737 INX H :Subprocessor 0438 Z3 02737 INX H :Subprocessor to inject current date 0437 E8 02738 PCHL :Subprocessor to inject sucent date 02740 CONDUT\$Date: ; subprocessor to inject sucent date 02741 CONDUT\$Set\$Forced\$Input: :Subprocessor to inject time into 02742 CONDUT\$Set\$Forced\$Input: :Subprocessor to set the date by taking 0444 23800F 02750 LXI H, Time\$In\$ASCII 0444 23990F 02750 LXI H, Time\$In\$ASCII 0445 13802 02757 UN :A the next 8 char			CONCLIT&Set \$Nor	nal:		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	0425 210002					•Set vector back to pormal
02730 ; 02732 CONOUT\$Match: 02732 CONOUT\$Match: 0438 23 02732 0430 23 02733 INX H 0430 23 02735 INX H 0432 23 02735 INX H 0435 E6 02737 XCHG ;HL -> subprocessor 0436 25 02738 PCHL ;Goto subprocessor 02740 CONUT\$Date: ;Subprocessor to inject current date 02741 ;Subprocessor to inject current date ;forced input) 0241 218F0F 02743 LXI H, Date 02740 CONUT\$Set\$Forced\$Input: CONUT\$Set\$Forced\$Input; 0444 0444 228D0F 02745 SHLD CEBForced\$Input; 0444 218P0F 02746 RET ;Subprocessor to inject time into 02744 CONUT\$Set\$Forced\$Input; ;Subprocessor to set the date by taking ; 02445 2190F 02757 LXI H, Time\$In\$ASCII ;Aateabacters of console output 04452						
$\begin{array}{c} 02731\\ 02732\\ 0430 \\ 0441 \\ 0430 \\ 0441 \\ 0430 \\ 0441 \\ 0441 \\ 0441 \\ 0441 \\ 0442 \\ 0441 \\ 0442 \\ 0441 \\ 0442 \\ 0441 \\ 0442 \\ 0441 \\ 0442 \\ 0441 \\ 0442 \\ 04$	0438 031504			CONDUTASETAPTOC	essor	; for subsequent characters
$\begin{array}{c c c c c c c c c c c c c c c c c c c $			7			
$\begin{array}{cccccccccccccccccccccccccccccccccccc$						
0430 25 02734 MOV E,M ;Get LS byte 0430 2335 INX H ;Get MS byte 0437 20735 INX H ;Get MS byte 0437 20736 NX HD ;Get MS byte 0440 E9 02737 XCHG ;HL ;Subprocessor 0441 E0 CONUT\$Date: ;Subprocessor into console input stream (using 02742 02740 CONUT\$Set\$Forced\$Input: forced input stream (using 02742 input stream (using 02743 0444 22800F 02745 SHLD CB#Forced\$Input: 0444 22800F 02746 RET ;Subprocessor to inject time into 02747 02747 ; CONOUT\$Time: ;Subprocessor to set the date by taking 02751 0448 C34040 02751 JMP CONUT\$Set\$Forced\$Input 0448 C34040 02752 ; Subprocessor to set the date by taking 02753 0445 1970F 02750 LXI H, Time\$In\$ASCII 04451 SE02 <td></td> <td>02732</td> <td>CONOUT\$Match:</td> <td></td> <td></td> <td></td>		02732	CONOUT\$Match:			
0430 25 02734 MOV E,M ;Get LS byte 0430 2335 INX H ;Get MS byte 0437 20735 INX H ;Get MS byte 0437 20736 NX HD ;Get MS byte 0440 E9 02737 XCHG ;HL ;Subprocessor 0441 E0 CONUT\$Date: ;Subprocessor into console input stream (using 02742 02740 CONUT\$Set\$Forced\$Input: forced input stream (using 02742 input stream (using 02743 0444 22800F 02745 SHLD CB#Forced\$Input: 0444 22800F 02746 RET ;Subprocessor to inject time into 02747 02747 ; CONOUT\$Time: ;Subprocessor to set the date by taking 02751 0448 C34040 02751 JMP CONUT\$Set\$Forced\$Input 0448 C34040 02752 ; Subprocessor to set the date by taking 02753 0445 1970F 02750 LXI H, Time\$In\$ASCII 04451 SE02 <td>043B 23</td> <td>02733</td> <td>INX</td> <td>н</td> <td></td> <td>;HL -> LS byte of subprocessor</td>	043B 23	02733	INX	н		;HL -> LS byte of subprocessor
0430 23 02735 INX H 0432 54 02736 MOV D,M ;Get MS byte 0440 E9 02737 XCHG ;HL -> subprocessor 02740 CONUT\$Date: ;Subprocessor to inject current date 02741 ;into console input stream (using 02742 LXI H,Date 02743 CONUT\$Set\$Forced\$Input: 0444 228D0F 02745 SHLD 02744 CONUT\$Set\$Forced\$Input: 0444 228D0F 02746 RET 02747 ; console input stream (using 0448 21990F 02750 LXI H,Time\$In\$ASCII 0448 21990F 02753 CONUT\$Set\$Forced\$Input ; console input stream 02753 CONUT\$Set\$Date: ; Subprocessor to set the date by taking 02754 UNUT A,Jate\$Set\$Forced\$Input ; and storing them in the date string 0448 C3404 02757 LXI H,Time\$Date\$Flags ; Set the acter count 0453 B6 02756 LXI H,Time\$Date\$Flags ; Set character count 0453 B6 02756 LXI H,Date ; Set character count </td <td>043C 5E</td> <td>02734</td> <td>MOV</td> <td>E.M</td> <td></td> <td>:Get LS byte</td>	043C 5E	02734	MOV	E.M		:Get LS byte
0435E602736MOVD, M;Get MS byte0437E802737XCHG;HL -> subprocessor0440E902738PCHL;Goto subprocessor0274002749;;Subprocessor to inject current date02741;porced input;into console input stream (using0274202743LXI H,Date0441218F0F02743LXI H,Date0444228D0F02745SHLD0447CONOUT\$Set\$Forced\$Input0447C20746RET02748CONOUT\$Time:;Subprocessor to inject time into0274902746RET02749CONOUT\$Time:;Console input stream044821990F02750LXI H,Time\$In\$ASCII0448C3404002751JMP0448C3404002752;02753CONOUT\$Set\$Forced\$Input0448C340400275504452180FCONOUT\$Set\$Forced\$Input0453B6002758044522757MVI A,Date\$Set0453B60027580454C2750LXI H,Time\$Date\$Flags0453B60027600454C2760UPC0455218F0F0456C27610457027610457027610457027630453C300402764;0454C27600455C27600456C2760045702768<	0420 22					,000 20 0,00
$\begin{array}{cccccccccccccccccccccccccccccccccccc$						Cot MC byto
0440 E90278 02789PCHL;Goto subprocessor02740CONOUT\$Date:;Subprocessor to inject current date0274102741;into console input stream (using0441 218F0F02742LXIH,Date02742CONOUT\$Set\$Forced\$Input:0444 228D0F02745SHLDCB\$Forced\$Input0447 C902746RET;Return to BIOS' caller0274902747;console input stream02749CONOUT\$Set\$Forced\$Input0448 21990F02750LXI02752jconsole input stream0448 C3440402751JMP02753CONOUT\$Set\$Forced\$Input02754jconsole input stream02755jadtesset02756LXI02757JMP02758CONOUT\$Set\$Forced\$Input02759MOV0453 B60027570445 21800F027560451 3E02027570453 B60027580453 B60027590455 3E08027600455 3E08027600455 3E08027600455 3E08027600455 21800F0276102764;02765Jup CONOUT\$Set\$String\$Pointer0453 21800F027620454 21400F027630455 3E08027600456 23604027600457 21870F027610457 21870F027640453 3E08027600454 77027710455 3E08027600				D, P		
02730 02740 02741 02742CONOUT\$Date: (Subprocessor to inject current date (sind console input stream (using (social input))0441 218F0F 0444 228D0F 0444 228D0F 0447 C9CONOUT\$Set\$Forced\$Input: SHLD 02745CONOUT\$Set\$Forced\$Input: sRtD 028Forced\$Input: (Subprocessor to inject time into (social input stream))0444 228D0F 0447 C902745 02745SHLD CDNOUT\$Set\$Forced\$Input: (Subprocessor to inject time into (social input stream))0448 21990F 0448 21990F02750 02753LXI UMP CONOUT\$Set\$Forced\$Input (social input stream))0448 C34404 02751 0275302751 (Social input Set\$Forced\$Input) (social input stream))0448 C34404 0275502752 (social input stream))0448 21990F 0275402755 (LXI H,Time\$In\$ASCII (social input stream))0448 2102 0275502757 (Social input stream))0448 210400 02755LXI H,Time\$In\$ASCII (social input stream))0448 210400 02755LXI H,Time\$In\$ASCII (social input stream))0448 21050 0275702757 (NVI A,Date\$Set Flags) (Set character count)0444 0453 2808 0455 3208 0455 320802760 (Sing MVI A,A (social input stream))0451 2102 02763 (social input stream)02761 (LXI H,Date (social input stream))0453 2806 0455 2180F 0456 0456 21430F02761 (LXI H,Date (social input stream))0454 2180F 0455 21430F 045502763 (Social input stream)0455 2180F 045502764 (Social input stream))0450 21430F 045602768 (Social input stream) <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
02740CONOUT\$Date:Subprocessor to inject current date02741into console input stream (using0441 218F0F02743LXI H,Date0444 228D0F02744CONOUT\$Set\$Forced\$Input:0444 228D0F02745SHLD CD\$Forced\$Input:0447 C902746RET02748CONOUT\$Set\$Forced\$Input:0448 21990F02750LXI H,Time\$In\$ASCII0448 C3440402751JMP CONOUT\$Set\$Forced\$Input02752;CONOUT\$Set\$Forced\$Input02753CONOUT\$Set\$Forced\$Input02754; Subprocessor to set the date by taking02755; Subprocessor to set the date by taking02756LXI H,Time\$Date\$Flags02757WI A,Date\$Set0445 2180F027570445 2180F027570453 B6027570454 7027590455 3E08027600455 3E08027610457 218F0F027610457 218F0F027610457 218F0F027610450 202767JMP0450 202768LXI H,Time\$Date\$Flags0450 20404227610450 2040227620450 2180F027610451 320F027610452 3208027610453 3208027610454 2360F027610455 3208027610456 23604027620457 218F0F027610457 218F0F027610458 268027700459 21640F027760450 21640F02770 <td>0440 E9</td> <td></td> <td>PCHL</td> <td></td> <td></td> <td>;Goto subprocessor</td>	0440 E9		PCHL			;Goto subprocessor
02741 02742 02743 02743 0441 218F0F 0444 228D0F 0444 228D0F 0444 228D0F 0447 C902745 02745 02746 02746 02747 02747 02748 0447 C9 02749 0448 21990F 0448 21900F 0448 21990F 0448 21900F 0451 3E02 02755 0454 77 0454 77 0455 21A30F 0455 3E08 0456 02756 02759 0470 0474 H, Date 0457 2180F 0456 21400F 02763 0456 21400F 02764 02764 02764 02764 02765 0450 21430F 0450 21430F <br< td=""><td></td><td></td><td>;</td><td></td><td></td><td></td></br<>			;			
02742; forced input)0441 218F0F02743LXI H, Date0244 228D0F02744CONOUT\$Set\$Forced\$Input:0447 C902745SHLD CB\$Forced\$Input:0447 C902746RET02747;Subprocessor to inject time into0274802749;0274902749;0274902751JMP CONOUT\$Set\$Forced\$Input0448 21990F02750LXI H, Time\$In\$ASCII0448 C340402751JMP CONOUT\$Set\$Forced\$Input02753CONOUT\$Set\$Date:; Subprocessor to set the date by taking0275402755JMP CONOUT\$Set\$Flags0445 C3404002755, and storing them in the date string0445 21A30F02756LXI H, Time\$Date\$Flags0445 326002757MVI A, Date\$Set0453 45002760MVI A, 80453 7702759MOV M, A0453 7702761LXI H, Date0454 C36C0402762JMP CONOUT\$Set\$String\$Pointer02764;console timestring0453 C30F02761LXI H, Time\$Date\$Flags0454 C36C0402762JMP CONOUT\$Set\$String\$Pointer0276502761LXI H, Time\$Date\$Flags0453 C30F02763;0454 C30F02764LXI H, Time\$Date\$Flags0455 21A30F02765LXI H, Time\$Date\$Flags0455 21A30F02763;0455 21A30F02764LXI H, Time\$Date\$Flags0456 2080027700RA H0452 21A30F02768 <td></td> <td>02740</td> <td>CONOUT\$Date:</td> <td></td> <td>;Subpro</td> <td>cessor to inject current date</td>		02740	CONOUT\$Date:		;Subpro	cessor to inject current date
02742; forced input)0441 218F0F02743LXI H, Date0274602744CONOUT\$Set\$Forced\$Input:0444 228D0F02745SHLD CB\$Forced\$Input:0447 C902746RET02747;; Subprocessor to inject time into0274802749; console input stream0448 21990F02750LXI H, Time\$In\$ASCII0448 C340402751JMP CCNOUT\$Set\$Forced\$Input0448 C340402753; the next 8 characters of console output0448 C340402755; the next 8 characters of console output0445 C340402755; and storing them in the date string0445 13E0202757MVI A,Date\$Set0453 3E0802760MVI A,80453 7702759MOV M.A0453 3E0802760MVI A,80454 7702763; the next 8 character count0455 3E0802760MVI A,80454 7702764; CONOUT\$Set\$Time:0455 3E0802760MVI A,80455 3E0802761LXI H, Date0455 21870F02761LXI H, Time\$Date\$Fing\$Fointer0276502761LXI H, Time\$Date\$Fing\$Gitter0455 21430F02765;0455 21430F027660455 21430F027670455 21430F027690455 21430F027690455 21430F027690455 21430F027690455 21430F027690455 21430F027690455 21430F027690455 21430F <td></td> <td>02741</td> <td></td> <td></td> <td>: into</td> <td>console input stream (using</td>		02741			: into	console input stream (using
0441 218F0F 02743 LXI H, Date 0444 228D0F 02745 CNOUT\$Set\$Forced\$Input; 0447 C9 02746 RET ;Return to BIOS' caller 02747 ;02748 CONOUT\$Set\$Forced\$Input; ;Subprocessor to inject time into 02749 ;02748 CONOUT\$Time: ;console input stream 0448 21990F 02750 LXI H, Time\$In\$ASCII 0448 C34404 02751 UMP CONOUT\$Set\$Forced\$Input 02754 ;202753 ;Subprocessor to set the date by taking 02755 ;Subprocessor to set the date by taking ;ste next 8 characters of console output 02754 ;2180F 02755 ;the next 8 characters of console output 02755 ;Ste flag to indicate that the ;and storing them in the date string 0451 3E02 02757 MVI A, Date\$Set\$; bat character count 0453 B6 02758 ORA H ;Set character count 0453 2180F 02761 LXI H, Date ;Set character count 0455 3E04 02762 JMP						
02744 CONDUT\$Set\$Forced\$Input: 0444 228D0F 02745 SHLD CB\$Forced\$Input 0447 C9 02746 RET ;Return to BIOS' caller 02748 CONDUT\$Ime: ;Subprocessor to inject time into 02749 ;console input stream 0448 21990F 02750 LXI 0448 C34404 02751 UMP CONOUT\$Set\$Forced\$Input 02753 CONOUT\$Set\$Date: ;Subprocessor to set the date by taking 02754 CONOUT\$Set\$Date: ;Subprocessor to set the date by taking 02755 CONOUT\$Set\$Date: ;subprocessor to set the date by taking 02755 CONOUT\$Set\$Date: ;subprocessor to set the date by taking 02755 CONOUT\$Set\$Date: ;subprocessor to set the date by taking 0445 277 02756 LXI 0453 SEO 02757 MVI 0455 268 02761 LXI 0455 278 DRA 0455 278 DRA 0455 <td< td=""><td>0441 218E0E</td><td></td><td>1 1 1</td><td>H Date</td><td>,</td><td>cu input/</td></td<>	0441 218E0E		1 1 1	H Date	,	cu input/
0444228DOF02745SHLDCB\$Forced\$Input0447C902746RET;Return to BIOS' caller0274802747;;Subprocessor to inject time into0274902748CONOUT\$Time:;console input stream044821990F02750LXIH,Time\$In\$ASCII0448C3440402751JMPCONOUT\$Set\$Forced\$Input02753cONOUT\$Set\$Date:;Subprocessor to set the date by taking0275402755;CONOUT\$Set\$Eate;subprocessor to set the date by taking02755cONOUT\$Set\$Date:;subprocessor to set the date by taking044821A3OF02756LXI04513E020275704522160MOV045326002758045502769MOV045502761LXI0457218FOF027610455218FOF027610457218FOF027610455218FOF027610455218FOF0276104560276402764;0276502764027660276402767; the next 8 characters of console output045021A3OF0453027680454202769045521A3OF04550276402764;027650276804502010045020100450201004502077005321A3OF <td>0111 2101 01</td> <td></td> <td></td> <td></td> <td></td> <td></td>	0111 2101 01					
0447 C902746RET;Return to BIOS' caller0274802747;; console input stream0488 21990F02750LXIH,Time\$In\$ASCII0448 C3440402751JMPCONOUT\$Set\$Forced\$Input02752;console input stream0448 C3440402751JMP02753CONOUT\$Set\$Date:; Subprocessor to set the date by taking02754; the next 8 characters of console output02755; and storing them in the date string044E 21430F02756LXI044E 21430F02757MVI0453 86002758ORA0453 7702759MOV0454 7702759MOV0455 218F0F02761LXI0454 7702762JMP0455 218F0F02761LXI02765CONOUT\$Set\$String\$Pointer02764;02765CONOUT\$Set\$Time:027660276702767JMP0450 21A30F027680450 21A30F027690450 21A30F027690450 21A30F027690453 77027690453 77027790454 3508027700453 7702779045436080453 7702779045436080453 7702770045436080453 7702771045436080453 7702771045436080453 77027710454 <td< td=""><td></td><td></td><td></td><td></td><td></td><td></td></td<>						
02747;02748CONDUT\$Time:02749;027490275002750LXI0448C340402751JMP0448C340402752;02753CONDUT\$Set\$Forced\$Input02754;02755;02755;02756LXI02757MVI02758CONDUT\$Set\$Eases044821920F02755;02756LXI02757MVI02758ORA0453B602759MOV0453SE002760MVI0453SE002761LXI0453SE0027630454;0276402765CONOUT\$Set\$Sting\$Pointer02763;0276402765CONOUT\$Set\$Time:02765;0276702768LXI02769MVI0276902769MVI0453277002769MVI045321430F02769MVI045321430F02769MVI045321430F02769MVI045321430F02769MVI045321430F02769MVI045321430F02769MVI045321430F02769MVI045321430F<				CB\$Forced\$Input		
02748 02749CONOUT\$Time:;Subprocessor to inject time into ; console input stream0448 21990F 0448 C3440402751LXIH,Time\$In\$ASCII0448 C3440402751JMPCONOUT\$Set\$Forced\$Input02752 02753CONOUT\$Set\$Date:;Subprocessor to set the date by taking 0275402754 02755console any taking 02755; the next 8 characters of console autput in the date string044E 21A30F02756LXIH,Time\$Date\$Flags0451 350202757MUIA,Date\$Set\$0453 86002758ORAM0453 7702769MOVM,A0455 218F0F02761LXIH,Date0454 77702769JMPCONOUT\$Set\$String\$Pointer0455 218F0F02761LXIH,Date02764 02763; 02764; 02765ithe next 8 characters of console output is and storing them in the time by taking 027660455 21A30F02768LXIH,Time\$Date\$Flags0450 21A30F02768LXIH,Time\$Date\$Flags0453 7702768LXIH,Time\$Date\$Flags0455 21A30F02768LXIH,Time\$Date\$Flags0450 21A30F02768LXIH,Time\$Date\$Flags0450 21A30F02769MVIA,Time\$Set0450 21A30F02769LXIH,Time\$Date\$Flags0450 21A30F02769LXIH,Time\$Date\$Flags0453 7702779MVIA,Time\$Set0454 360802770ORAM0452 86<	0447 C9		RET		;Return	i to BIOS' caller
02749; console input stream0448 21990F02750LXIH, Time\$In\$ASCII044B C3440402751JMPCONOUT\$Set\$Forced\$Input02752;CONOUT\$Set\$Date:; subprocessor to set the date by taking0275302754; the next 8 characters of console output02754; the next 8 characters of console output02755; the next 8 characters of console output044E 21A30F02756LXI0451 3E0202757MVI0453 B602758ORA0453 3E0802760MVI0455 3E0802760MVI0455 3E0802760MVI0455 3E0802760MVI0455 3E0802760MVI0456 17702762JMP0457 218F0F02761LXI02764;02765CONOUT\$Set\$String\$Pointer02766;02767;0450 21430F027680451 2430F027680450 21430F027680450 21430F027690450 21430F027680451 21430F027690450 3201027690450 21430F027690450 3201 </td <td></td> <td></td> <td>;</td> <td></td> <td></td> <td></td>			;			
044821990F02750LXIH,Time\$In\$ASCII044BC3751JMPCONOUT\$Set\$Forced\$Input02752;CONOUT\$Set\$Forced\$Input02753cONOUT\$Set\$Date:; Subprocessor to set the date by taking02754; the next 8 characters of console output02755; the next 8 characters of console output02756LXIH,Time\$Date\$Flags044E21A30F0275604513E02027570453D64027580453045602759045302760MVI04533E08027600457218F0F027610457218F0F027610457218F0F0276102764;02765CONOUT\$Set\$Stime;02766cONOUT\$Set\$Stime;02767; the next 8 characters of console output02764;027650276402765CONOUT\$Set\$Time;027660276402767; the next 8 characters of console output02767; the next 8 characters of console output045321A30F02768LXI0450220102769MVI04537702769MVI04537702769MVI04537702770QRA045321A30F02771MOV04537702771MOV04543E0802773LXI <td></td> <td></td> <td>CONOUT\$Time:</td> <td></td> <td>;Subpro</td> <td>cessor to inject time into</td>			CONOUT\$Time:		;Subpro	cessor to inject time into
044821990F02750LXIH,Time\$In\$ASCII044BC3404002751JMPCONOUT\$Set\$Forced\$Input02752;02753; Subprocessor to set the date by taking02754		02749			; cons	ole input stream
044B C3440402751JMPCONOUT\$Set\$Forced\$Input02752;;Subprocessor to set the date by taking02753;the next 8 characters of console output02755;and storing them in the date string044E 21A30F02756LXIH,Time\$Date\$Flags0451 3E0202757MVIA,Date\$Set0453 7702759MOVM,A0455 3E0802760MVIA,80455 3E0802761LXIH,Date0455 3E0802762JMPCONOUT\$Set\$String\$Pointer0453 7702761LXIH,Date0454 7702762JMPCONOUT\$Set\$String\$Pointer0455 3E0802760MVIA,80457 218FOF02761LXIH,Date02764;;the next 8 character count02764;;;02765;;02766;;02767WIA,Time\$Date\$Flags0450 21A30F02768LXI0463 260102769MVI0463 7702771MOV0464 360802772MVI0465 2190F02773LXI0466 21990F02773LXI0467 2180P02773LXI0468 2190P02773LXI0469 C36C0402774JMP0469 C36C0402775;0469 C36C040277402775;0469 C36C040277402775;0469 C3	0448 21990F	02750	LXI	H.Time\$In\$ASCII		
02752;02753CONOUT\$Set\$Date:; Subprocessor to set the date by taking02754; the next 8 characters of console output02755; and storing them in the date string04513E020275704513E02027570453B6027580453B60275804533E02027570453B6027580453B60275804533E0802760045470275904553E080276004553E080276004551LXI04552761LXI0457218F0F02763;02764;02765CONOUT\$Set\$Time:02766;02767;02768LXI02769MVI04503E0102769MVI045237700780MVI04537702770027640454377027700276404643E0802772MVI0452190F02773LXIH, Time\$In\$ASCII046521990F02774JMP046602767027740467046821990F0277302774046904690460027740460027740461<	044B C34404	02751	. IMP	CONDUT\$Set\$Forc	edsinout	
02753CONOUT\$Set\$Date:;Subprocessor to set the date by taking 0275402754;the next 8 characters of console output 22755;the next 8 characters of console output 3 and storing them in the date string044E 21A30F02756LXIH,Time\$Date\$Flags;Set flag to indicate that the0451 3E0202757MUXA,Date\$Set ;date has been set by program0453 8602758ORAM0453 3E0802760MUVM,A0455 3E0802761LXIH,Date0457 218F0F02761LXIH,Date02763;02764;Subprocessor to set the time by taking 0276502764;CONOUT\$Set\$Stime:;Subprocessor to set the time by taking 0276602763;;;02764;;Subprocessor to set the time by taking 0276602765Q2768LXIH,Time\$Date\$Flags0450 3E0102769MVIA,Time\$Set ;0452 21A30F02768LXIH,Time\$Date\$Flags0453 7702770QRAM0453 7702771MOVM,A0454 3E0802772MVIA,80454 3E0802773LXIH,Time\$In\$ASCII0455 7702771MVIA,80454 3E0802772MVI0455 7702771MVI0454 3E080277202775;0459 C36C040277302774JMP0459 C36C040277402775; </td <td>0112 001101</td> <td></td> <td></td> <td></td> <td>cutinput</td> <td></td>	0112 001101				cutinput	
02754; the next 8 characters of console output output; and storing them in the date string044E 21A30F02756LXIH,Time\$Date\$Flags ; Set flag to indicate that the0451 3E0202757MVIA,Date\$Set; date has been set by program0453 8602758ORAM0455 3E0802760MVIA,8; Set character count0455 3E0802760MVIA,8; Set character count0457 218FOF02761LXIH,Date; Set address0457 218FOF02762JMPCONOUT\$Set\$String\$Pointer02763; 02765; concerts; subprocessor to set the time by taking 0276602764; 02765; concerts; ad storing them in the time string0450 21A30F02768LXIH,Time\$Date\$Flags; Set flag to indicate that the0463 3E0102769MVIA,Time\$Set; time has been set by program0463 7702771MOVM,A0464 3E0802772WIA,8;Set character count0464 21990F02773LXIH,Time\$In\$ASCII;Set address0469 C36C0402774JMPCONOUT\$Set\$String\$Pointer0459 C36C0402773LXIH,Time\$in\$ASCII;Set address0469 C36C0402774JMPCONOUT\$Set\$String\$Pointer					- Cubau	anness be and the date by believe
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$			CONDOTASELADALI	21		
044E 21A30F02756LXIH,Time\$Date\$Flags;Set flag to indicate that the0451 3E0202757MVIA,Date\$Set; date has been set by program0453 86027580RAM0454 7702759MOVM,A0453 3E0802760MVIA,80453 3E0802761LXIH,Date0457 218FOF02761LXIH,Date0457 218FOF02762JMPCONOUT\$Set\$String\$Pointer02763;02764;02764;;02765CONOUT\$Set\$Time:; Subprocessor to set the time by taking02764;;0276502768LXI0450 21A30F02768LXI0452 820102767; and storing them in the time string0463 7702770ORA0463 820802771MOV0464 3E0802772MVI0466 21990F02773LXI047602774JMP0469 C36C040277402775;						
0451 3E02 02757 MVI A,Date\$Set ; date has been set by program 0453 3E6 02758 ORA M 0455 3E6 02758 ORA M 0455 3E08 02760 MVI A,8 0455 3E08 02760 MVI A,8 0455 3E08 02760 MVI A,8 0457 218FOF 02761 LXI H,Date 0456 C36C04 02762 JMP CONOUT\$Set\$String\$Pointer 02763 ; 02764 ; 02764 ; the next 8 characters of console output 02765 CONOUT\$Set\$Time: ; subprocessor to set the time by taking 02766 ; the next 8 characters of console output 02767 ; and storing them in the time string 0450 3E01 02769 MVI 0450 3E01 02769 MVI 0452 377 02771 MOV 0453 376 02771 MOV 0464 21990F 02773 LXI 0456 21990F 02773 LXI 0456 21990F 02774 JMP						
0453 B6 02758 ORA M 0454 77 02759 MOV M,A 0455 3E08 02760 MVI A,8 ;Set character count 0457 218FOF 02761 LXI H,Date ;Set address 0456 C36C04 02763 ; 02764 ; 02765 ;the next 8 characters of console output 02765 (CONOUT\$Set\$Time: ;Subprocessor to set the time by taking 02765 ;the next 8 characters of console output ; and storing them in the time string 0450 3E01 02769 MVI A,Time\$Set ; time has been set by program 0463 77 02771 MOV M,A 0464 3E08 02773 LXI H,Time\$IntaKSCII ;Set character count 0465 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer					95	
0453 B6 02758 ORA M 0454 77 02759 MOV M,A 0455 3E08 02760 MVI A,8 ;Set character count 0457 218F0F 02761 LXI H,Date ;Set address 0456 C36C04 02762 JMP CONOUT\$Set\$String\$Pointer 02763 ; 02764 ; 02765 CONOUT\$Set\$Time: ;Subprocessor to set the time by taking 02765 cONOUT\$Set\$Time: ;Subprocessor to set the time by taking 02765 cONOUT\$Set\$Time: ;Subprocessor to set the time by taking 02766 ; the next 8 characters of console output 02767 ; the next 8 characters of console output 02768 LXI H,Time\$Date\$Flags ;Set flag to indicate that the 0460 3E01 02769 MVI A,Time\$Set ; time has been set by program 0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 ;Set character count 0464 3E08 02773 LXI H,Time\$in\$ASCII ;Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;		02757				; date has been set by program
0454 77 02759 MOV M,A 0455 3E08 02760 MVI A,8 ;Set character count 0457 218FOF 02761 LXI H,Date ;Set address 0458 C36C04 02762 JMP CONOUT\$Set\$String\$Pointer 02763 ; 02765 CONOUT\$Set\$Time: ;Subprocessor to set the time by taking 02766 ; the next 8 characters of console output 02767 ; and storing them in the time string 0450 21A30F 02768 LXI H,Time\$Date\$Flags ;Set flag to indicate that the 0460 3201 02769 MVI A, Time\$Date\$Flags ;Set flag to indicate that the 0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 ;Set character count 0466 21990F 02773 LXI H,Time\$Date\$Flags ;Set character count 0466 21990F 02773 LXI H,Time\$Date\$Flags ;Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;	0453 B6	02758	ORA			
0455 3E08 02760 MVI A,8 ;Set character count 0457 218F0F 02761 LXI H,Date ;Set address 045A C36C04 02762 JMP CONOUT\$Set\$String\$Pointer 02763 ; 02764 ; 02765 CONOUT\$Set\$Time: ;Subprocessor to set the time by taking 02765 CONOUT\$Set\$Time: ;Subprocessor to set the time by taking 02766 ; the next 8 characters of console output 02767 ; and storing them in the time string 0450 21A30F 02768 LXI H,Time\$Date\$Flags ;Set flag to indicate that the 0460 3E01 02769 MVI A,Time\$Set ; time has been set by program 0462 377 02771 MOV M,A 0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 ;Set character count 0464 21990F 02773 LXI H,Time\$in\$ASCII ;Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;	0454 77		MOV	M.A		
0457 218F0F 02761 LXI H, Date ; Set address 045A C36C04 02762 JMP CONDUT\$Set\$String\$Pointer 02763 ; 02764 ; 02764 ; 02765 CONDUT\$Set\$String\$Pointer 02765 cONOUT\$Set\$Time: ; Subprocessor to set the time by taking 02766 ; the next 8 characters of console output ; and storing them in the time string 045D 21A30F 02768 LXI H, Time\$Date\$Flags ; Set flag to indicate that the 0450 2E01 02769 MVI A, Time\$Date\$Flags ; time has been set by program 0463 2E01 02770 ORA M 0463 3E08 02772 MVI A, 8 ; Set character count 0464 3E08 02772 MVI A, 8 ; Set character count 0464 21990F 02773 LXI H, Time\$in\$ASCII ; Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ; Set address						:Set character count
045A C36C04 02762 JMP CONOUT\$Set\$String\$Pointer 02763 ;						
02763 ; 02764 ; 02765 ; 02766 ; 02767 ; 02767 ; 02767 ; 02767 ; 02767 ; 02767 ; 0450 22767 0450 22769 0460 3E01 02769 MVI 0462 86 02770 QRA 0463 77 02771 MOV 0463 21990F 02773 LXI LXI H, Time\$in\$ASCII 9469 C36C04 02773 LXI H, Time\$in\$ASCII 0469 C36C04 02774 JMP 02775 ;						
02764 ; 02765 CONOUT\$Set\$Time: ; Subprocessor to set the time by taking 02765 ; the next 8 characters of console output 02766 ; and storing them in the time string 045D 21A30F 02768 LXI 0450 21A30F 02768 LXI 0450 21A30F 02769 MVI 0450 21A30F 02769 MVI 0463 2E01 02770 ORA 0463 77 02771 MOV 0463 3E08 02772 MVI 0464 3E08 02773 LXI 0466 21990F 02773 LXI 0469 C36C04 02774 JMP 02469 C36C04 02774 JMP 02775 ;	045A C36C04			CONDUT#Set#Str1	ngaroint	.er
02765 CONOUT\$Set\$Time: ; Subprocessor to set the time by taking 02766 ; the next 8 characters of console output 02767 ; and storing them in the time string 0450 02768 LXI 0450 02769 MVI 0460 3E01 02770 0452 02770 ORA 0463 77 02771 0464 3E08 02772 0465 1990F 02773 0466 21990F 02773 0469 02774 JMP 0469 02774 JMP 0469 02774 JMP 0469 02775 ;			;			
02766 ; the next 8 characters of console output 02767 ; and storing them in the time string 045D 21A30F 02768 LXI H,Time\$Date\$Flags ; set flag to indicate that the 0460 3E01 02769 MVI A,Time\$Set ; time has been set by program 0462 86 02770 ORA M 0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 ; Set character count 0464 21990F 02773 LXI H,Time\$in\$ASCII ; Set address 0469 C36C04 02774 JMP CONDUT\$set\$string\$Pointer 02775 ;			;			
02767 ; and storing them in the time string 045D 21A30F 02768 LXI H,Time\$Date\$Flags ; Set flag to indicate that the 0450 2601 02769 MVI A,Time\$Date\$Flags ; Set flag to indicate that the 0460 3E01 02770 MVI A,Time\$Date\$Flags ; time has been set by program 0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 0464 3E08 02773 LXI H,Time\$in\$ASCII ; Set character count 0464 21990F 02773 LXI H,Time\$in\$ASCII ; Set address 0469 C36C04 02774 JMP CONDUT\$Set\$String\$Pointer 02775 ;		02765	CONOUT\$Set\$Time	5:	;Subpro	cessor to set the time by taking
045D 21A3OF 02768 LXI H,Time\$Date\$Flags ;Set flag to indicate that the 0460 3E01 02769 MVI A,Time\$Date\$Flags ;Set flag to indicate that the 0460 3E01 02769 MVI A,Time\$Set ;time has been set by program 0462 86 02770 ORA M 0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 ;Set character count 0466 21990F 02773 LXI H,Time\$in\$ASCII ;Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;		02766			; the	next 8 characters of console output
045D 21A3OF 02768 LXI H,Time\$Date\$Flags ;Set flag to indicate that the 0460 3E01 02769 MVI A,Time\$Date\$Flags ;Set flag to indicate that the 0460 3E01 02769 MVI A,Time\$Set ;time has been set by program 0462 86 02770 ORA M 0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 ;Set character count 0466 21990F 02773 LXI H,Time\$in\$ASCII ;Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;					; and	storing them in the time string
0460 3E01 02769 MVI A,Time\$Set ; time has been set by program 0462 B6 02770 ORA M 0462 B6 02770 ORA M 0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 ; Set character count 0466 21990F 02773 LXI H,Time\$in\$ASCII ; Set address 0469 C36C04 02774 JMP CONOUT\$Set\$string\$Pointer 02775 ; ;	045D 21A30F		LXI	H.Time\$Date\$Fla	95	:Set flag to indicate that the
0462 B6 02770 ORA M 0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 ;Set character count 0464 21990F 02773 LXI H,Time\$in\$ASCII ;Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;						
0463 77 02771 MOV M,A 0464 3E08 02772 MVI A,8 ;Set character count 0466 21990F 02773 LXI H,Time\$in\$ASCII ;Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;						, time has been set by program
0464 3E08 02772 MVI A,8 ;Set character count 0466 21990F 02773 LXI H,Time\$in\$ASCII ;Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;						
0466 21990F 02773 LXI H,Time\$in\$ASCII ;Set address 0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;	0463 //					.
0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;						
0469 C36C04 02774 JMP CONOUT\$Set\$String\$Pointer 02775 ;	0466 21990F	02773	LXI	H,Time\$in\$ASCII		;Set address
02775 ;		02774	JMP	CONOUT\$Set\$Stri	ng\$Point	er
		02776		ing\$Pointer.		;HL \rightarrow string, A = count
027/7 CUNDOT#Set#String#Former: ;nc -/ string, A - Count 046C 32D003 02777 STA CONOUT#String#Length ;Save count	0440 330003		CONCOLAGE LAGEL.	CONDUTES	anath	
046E-32D003-02777 STA CUNOUI®String®Length ;Save count 046E-32DE03-02778 SHLD CONOUI\$String®Pointer :Save address	0460 320003		514	CONDUT#String%L	engen	
	046F 22CE03					
0472 217804 02779 LXI H,CONOUT\$Process\$String ;Vector further output	0472 217804	02779	LXI	H,CONOUT\$Proces	s\$String	; ;Vector further output

.

254 The CP/M Programmer's Handbook

0475 C31504	02780		JMP	CONOUT\$Set\$Proc	essor	
	02781	; CONOUT\$P	rocecet	String:	Contro	l arrives here for each character
	02783	CONCOTAR	rucess#.	517 ING.		he string in register C. The
	02784					acters are stacked into the
	02785					iving string until either a 00-byte
	02786				; is e	ncountered or the specified number
	02787					haracters is stacked.
0478 2ACE03	02788		LHLD	CONOUT\$String\$P		;Get current address for stacking chars
047B 79	02789		MOV	A,C		;Check if current character is OOH
047C B7	02790		ORA	A		
047D CA3504	02791		JZ	CONOUT\$Set\$Norm	al	Revert to normal processing
0480 77	02792		Mav	M, A		Otherwise, stack character
0481 23	02793		INX	н		;Update pointer
0482 3600	02794		MVI	M, 00H		Stack fail-safe terminator
0484 22CE03	02795		SHLD	CONCUT\$String\$F	ointer	;Save updated pointer
0487 21D003	02796		LXI	H,CONOUT\$String	≸Length	;Downdate count
048A 35	02797		DCR	м		
048B CA3504	02798		JZ	CONOUT\$Set\$Nor#	al	;Revert to normal processing
	02799		-			; if count hits O
048E C9	02800		RET			Return with output vectored back;
	02801					; to CONOUT\$Process\$String
	02802	;				1
		;#				
	02901	;				
	02902	;	Auxilia	ry input status		
	02903	;				
	02904			utine checks the		er count in the
	02905			iate/input_buffe		
	02906				o indica	te whether or not
	02907	•	data is	waiting.		
	02908	;				
	02909		Entry p	arameters: none.		
	02910	;				
	02911	;	Exit pa	rameters		
	02912	;				
	02913	;		A = 000H if the	re is no	data waiting
	02914	;		A = OFFH if the	ere 15 da	ta waiting
	02915 02916	;				
	02918	AUXIST:			•/=== B	IOS entry point (Private)
	02918				,	ios entry point (hirdate)
048F 2A5C00	02919			CB\$Auxiliary\$Ir	nut.	;Get redirection word
0492 116400	02920		LXI			sses ; and table pointer
0495 CD6F06	02921		CALL			;Get device table address
0498 C34708	02922		JMP	Get\$Input\$Statu		;Get status from input device
0170 001700	02923				-	; and return to caller
	02924					
	03000	;#				
	03001	;				
	03002		Auxilia	ry output status	5	
	03003	;				
	03004		This ro	utine sets the A	registe	r to indicate whether the
	03005					y to accept output data.
	03006					used for auxiliary output, this
	03007					f all of their statuses.
	03008	;				
	03009	;	Entry p	arameters: none		
	03010	;				
	03011		Exit pa	rameters		
	03012	;				
	03013	;				list devices are not ready
						vices are ready
	03014	,				
	03015	;				
	03015 03016	;				
	03015 03016 03017	; ; ;=======				
	03015 03016 03017 03018	; ; ;=================================			;<=== B	IOS entry point (Private)
	03015 03016 03017 03018 03018	; ; ====== AUXOST: ; ======				
	03015 03016 03017 03018 03019 03020	; ;====== AUXOST: ;======	ssesses LHLD	 CB\$Auxiliary\$Ou	itput	IOS entry point (Private) ;Get list redirection word
	03015 03016 03017 03018 03019 03020 03021	; ; ;======= AUXOST: ;======			itput	
	03015 03016 03017 03018 03019 03020 03021 03022	; ; ;====== AUXOST: ;======	ssesses LHLD	 CB\$Auxiliary\$Ou	itput	
	03015 03016 03017 03018 03019 03020 03021 03022 03100	; ; ;======= AUXOST: ;======	ssesses LHLD	 CB\$Auxiliary\$Ou	itput	
	03015 03016 03017 03018 03019 03020 03021 03022 03100 03101	; ; AUXOST: ;======	LHLD JMP	CB\$Auxiliary\$Ou Get\$Composite\$S	itput Status	;Get list redirection word
049B 2A5E00 049E C37905	03015 03016 03017 03018 03019 03020 03021 03022 03100 03101 03102	; ;======= AUXOST: ;======= ; ;# ;	LHLD JMP	 CB\$Auxiliary\$Ou	itput Status	;Get list redirection word
	03015 03016 03017 03018 03019 03020 03021 03022 03100 03101	; ;======= AUXOST: ;======= ; ;# ;	ELLD JMP Auxilia:	CB\$Auxiliary\$Ou Get\$Composite\$S ry input (replac	itput Status Sement fo	;Get list redirection word

			;	appropriate logical auxiliary device.
		03106 03107	;	
		03107	;	Entry parameters: none.
		03109	;	Exit parameters
		03110	;	
		03111	;	A = data character
		03112 03113	;	
			AUXIN:	;<=== BIOS entry point (standard)
		03115		
	2A5C00	03116		LHLD CB\$Auxiliary\$Input ;Get redirection word
0444	116400	03117		LXI D, CB\$Device\$Table\$Addresses ; and table pointer
	CD6F06 C39106	03118 03119		CALL Select\$Device\$Table ;Get device table address JMP Get\$Input\$Character ;Get next input character
0100	037108	03120		; and return to caller
		03121	;	
		03200		
		03201 03202	;	Auxiliary output (replaces PUNCH)
		03202	,	This routine outputs a data byte to the auxiliary device(s).
		03204		It is similar to CONOUT except that it uses the watchdog
		03205	;	timer to detect if a device stays busy for more than
		03206	;	30 seconds at a time. It outputs a message to the console
1		03207 03208	:	if this happens.
1		03208		Entry parameters
1			;	
		03211	;	C = data byte
0.000		03212	;	
04AU	OD0A07413	/503213 03214	AUXUUT\$	Busy\$Message: DB CR,LF,7,^Auxiliary device not Ready?^,CR,LF,O
		03215	,	
		03216	AUXOUT:	
		03217		
	2A5E00 11AD04	03218		LHLD CB\$Auxiliary\$Output ;Get aux. redirection word
0401	TTADO4	03219 03220		LXI D,AUXOUT\$Busy\$Message ;Message to be output if time ; runs out
0404	C3A205	03221		JMP Multiple\$Output\$Byte
			;	
		03300		
			7	List status
			;	
		03304	;	This routine sets the A register to indicate whether the
r		03305		List Device(s) is/are ready to accept output data.
		03306		As more than one device can be used for list output, this
1		03307 03308		routine returns a Boolean AND of all of their statuses.
1				Entry parameters: none
1		03310	;	
		03311		Exit parameters
1		03312 03313	;	A = 000H if and an move list devices and and words
1		03313	; ;	A = 000H if one or more list devices are not ready A = 0FFH if all list devices are ready
1		03315	;	
1		03316	;	
1		03317		
l l		03318 03319	LISTST:	;<=== BIOS entry point (standard)
0407	246200	03319	,	LHLD CB\$List\$Output ;Get list redirection word
	C37905	03321		LHLD CB\$List\$Output ;Get list redirection word JMP Get\$Composite\$Status
1		03322	;	
		03400		التحق وبنقصية
1		03401 03402	;	List output
1		03403	;	This routine outputs a data byte to the list device.
		03404	;	It is similar to CONOUT except that it uses the watchdog
1		03405	7	timer to detect if the printer stays busy for more
1		03406 03407	;	than 30 seconds at a time. It outputs a message to the console if this happens.
		03407	;	11 (1115 Happens.
1		03409		Entry parameters
		03410	;	
1		03411	;	C = data byte
		03412	;	
L				

0400	0D0A0750	03414	LIST\$Busy\$M ;	-	DB	CR,LF,	,7,^Printer not Ready?^,CR,LF,O	
		03415	;=========		-			
		03416	LIST:			;<===	BIOS entry point (standard)	
OAFE	2A6200	03417 03418	;=========				;Get list redirection word	
	11DD04	03418			st\$Qutput [\$Busy\$Me		;Get list redirection word ;Message to be output if time	
J4F0	110004	03417		0,0101	*6057 #He	ssage	; runs out	
04FB	C3A205	03421	JMP	Multip	ole\$Outpu	t\$Byte	y rans dat	
		03422	;					
		03500	;#					
		03501	; Req	lest use r d	choice			
		03502	;					
		03503	; Thi	s routine c	lisplays	an errom	r message, requesting	
		03504	; ac	noice of:				
		03505	;					
		03506	;				ion that caused the error	
		03507	;				and attempt to continue	
		03508 03509	; .	A A	loort the	program	n and return to CP/M	
		03510		s routine =	ocente a	obarao	ter from the console,	
		03511	; con	verts it tr	n Hoperca	se and i	returns to the caller	
		03512		h the respo				
		03513	, ,				- /	
		03514	RUC\$Message	:				
	ODOA	03515		DB	CR,LF			
500	202020202			DB	1	Enter R	- Retry, I - Ignore, A - Abort : 1,0)
		03517	;					
		03518	;					
		03519	Request\$Use	r\$Choice:			October and the stand	
	CD2D03	03520	CAL			.	;Gobble up any type-ahead	
532	CA3B05	03521	JZ CAL		iffer\$Emp	ty		
	CD3A03 C32F05	03522 03523	JMP		st\$User\$C	hoigo		
538	C32F05	03523	JULE	Kednes	, touser ac	noice		
		03525	RUC\$Buffer\$	Emptys				
53B	21FE04	03526	LXI		Message		;Display prompt	
	CD5305	03527	CAL		\$Error\$M	essage		
		03528	2112					
541	CD3A03	03529	CALI				;Get console character	
544	CD3BOE 32B00D	03530	CAL				Make uppercase for comparisons	
547	32B00D	03531	STA		Action\$Co	nfirm	;Save in confirmatory message	
54A	F5	03532	PUS	H PSW			;Save for later	
		03533	LXI					
	21B00D	03534 03535	CAL		<pre>\$Action\$ \$Error\$M</pre>			
04E	CD5305	03536	UAL		. PETTOT PH	essage		
551	E 1	03537	POP	PSW			;Recover action code	
552		03538	RET	1.04			, Mecover action code	
		03539	:					
		03600	;#					
		03601	;					
		03602		out error m	nessage			
		03603	;					
		03604					message to all the currently	
		03605	; sel	ected consc	le devic	es excep	ot those being used to receive	
		03606					to avoid "deadly embrace" situations	
		03607					y for too long causes an error messag	je
		03608				nsole ou	stput is being directed to the	
		03609		nter as wel	. 1 •			
		03610 03611	; 		a makaa	ura of ·	nost of the CONCUT subrouting	
		03611					nost of the CONOUT subroutine. CONOUT using a private	
		03612		ry point.	Jugary 10	enterst	London disting a private	
		03614	; enc	, point.				
		03615		ry paramete	ers			
		03616	; בווכ	, paramete				
		03617	;	HL ->	00-byte	termina	ted error message	
		03618	;					
		03619	, Output\$Erro	r\$Message:				
553	E5	03620	PUS				;Save message address	
	2A5A00	03621	LHL	D CB\$Cor	nsole\$Out	put	;Get console redirection bit map	
1004		03622	XCH					
0557								
557	2A6200	03623	LHL	D CB\$Lis	st\$Output		;Get list redirection bit map	
557		03623 0362 4 03625	LHL	D CB\$Lis	st\$Output		;Get list redirection bit map ;HL = list, DE = console ;Now set to 0 all bits in the cons	

Figure 8-10. (Continued)

	03626			; bit map that are set to 1 in the \sim
	03627			; list bit map
055B 7C	03628	MOV	А,Н	;Get MS byte of list
055C 2F	03629	CMA	8,0	; Invert
			-	
055D A2	03630	ANA	D	Preserve only bits with O's
055E 67	03631	MOV	H,A	;Save result
055F 7D	03632	MOV	A,L	Repeat for LS byte of list;
0560 2F	03633	CMA		
0561 A3	03634	ANA	E	
0562 6F	03635	MOV	L,A	;HL now has only pure console
0002 0	03636		2711	; devices
05/0 04		ORA	н	
0563 B4	03637 03638	JZ	OEM\$Device\$Prese	;Ensure that at least one device nt ; is selected
0564 CA6A05				
0567 210100	03639	LXI	H, 0001H	Otherwise use default of device O;
	03640	OEM\$Device\$Pre		
	03641	OEM\$Next\$Chara	acter:	
056A D1	03642	POP	D	Recover message address into DE;
056B 1A	03643	LDAX	D	;Get next byte of message
056C 13	03644	INX	D	;Update message pointer
056D B7	03645	ORA	Ā	;Check if end of message
0360 8/			A	
056E C8	03646	RZ	_	;Yes, exit
056F D5	03647	PUSH	D	;Save message address for later
0570 E5	03648	PUSH	н 、	;Save special bit map
	03649			;Data character is in A
0571 CDD103	03650	CALL	CONOUT\$OEM\$Entry	Enter shared code
0574 E1	03651	POP	н	Recover special bit map
0575 C36A05	03652	JMP	0EM\$Next\$Charact	
	03653		content action at act	• '
	03654	;		
		•		
	03655	;		
	03656	; Get co	omposite status	
-	03657	;		
	03658	; Thisr	outine sets the A	register to indicate whether the
	03659		device(s) is/are	ready to accept output data.
	03660			can be used for output, this
	03661			n AND of all of their statuses.
	03662	:		and of all of their statusest
	03663		parameters	
	03664		Parameters	
		;		
	03665	;	HL = I/O redirec	tion bit map for output device(s)
	03666	;		
	03667	; Exitp	arameters	
	03668	;		
	03669	;	A = 000H if one	or more list devices are not ready
	03670	•		list devices are ready
	03671			
0578 00	03672	GCS\$Status:	DB 0	Composite status of all devices
00/0 00	03673	:	80 0	composite status of all devices
		*		
	03674			
	03675	Get\$Composite\$		
0579 3EFF	03676	MVI		Assume all devices are ready;
057B 327805	03677	STA	GCS\$Status	Preset composite status byte;
	03678			
057E 116400	03679	LXI	D,CB\$Device\$Tabl	Addresses ;Addresses of dev. tables
0581 D5	03680	PUSH	D	;Put onto stack ready for loop
0582 E5	03681	PUSH	Ĥ	;Save bit map
	03682	GCS\$Next\$Devic		, ort map
0583 E1		POP	e: H	· Deserves sendinged :
	03683			Recover redirection bit map
0584 D1	03684	POP	D	Recover device table addresses pointer
0585 CD6F06	03685	CALL	Select\$Device\$Tal	
0588 B7	03686	ORA	A	;Check if a device has been
	03687			; selected (i.e. bit map not all zero)
0589 CA9905	03688	JZ	GCS\$Exit	;No, exit
058C C5	03689	PUSH	B;Yes-B	
058D E5	03690	PUSH	H , I I I	;Save device table addresses pointer
058E CDOF06	03691	CALL	Check\$Output\$Rea	
0591 217805	03692	LXI	H,GCS\$Status	;AND together with previous devices
0594 A6	03692	ANA	M	
				; status
0595 77	03694	MOV	M, A	;Save composite status
	03695			
0596 C38305	03696	JMP	GCS\$Next\$Device	;Loop back for next device
	03697	;		
	03698	GCS\$Exit:		
0599 3A7805	03699	LDA	GCS\$Status	;Return with composite status
059C B7	03700	ORA	A	
0590 C9	03701	RET		
0070 07	03/01	NE I		
L				

	03702					
	03800	; ;#				
	03801	;				
	03802	; Multipl	e outpu	t byte		
	03803	;				
	03804	; This ro	outine o	utputs a	data by	te to the all of the
	03805					edirection word.
	03806 03807					that it uses the watchdog devices stays busy for more
	03808					outputs a message to the console
	03809		happen			delpets a message to the console
	03810	1				
	03811	; Entry F	arameter	rs		
	03812	;				
	03813	;		/O redire		
	03814	;		Message ta byte	to be ou	tput if time runs out
	03815 03816	:	c – ua	ta byte		
0708 =	03817	, MOB\$Maximum\$Bus		EQU	1800	Number of clock ticks (each at
0700 -	03818	///////////////////////////////////////	- /	200	1000	; 16.666 milliseconds) for which the
	03819					; device might be busy
059E 00	03820	MOB\$Character:		DB	0	;Character to be output
059F 0000	03821	MOB\$Busy\$Messag	je:	DW	0	Address of message to be
	03822	NORTH			٥	; output if time runs out
05A1 00	03823 03824	MOB\$Need\$Messag	le:	DB	U	;Flag used to detect that the ; watchdog timer timed out
	03824	•				, watchdog timer timed odt
	03826	, Multiple\$Output	t\$Byte:			
05A2 79	03827	MOV	A,C			;Get data byte
05A3 320807	03828	STA	MOB\$Ma	ximum\$Bu	SY	;Save copy
05A6 EB	03829	XCHG				;HL -> timeout message
05A7 229F05	03830	SHLD	MOB\$Bu	sy\$Messa	ge	;Save for later use
05AA EB	03831	XCHG				;HL = bit map again
05AB 116400	03832 03833	LXI		evice\$Ta	hla\$Addr	esses ;Addresses of dev. tables
05AE D5	03834	PUSH	D,0040	evicepia	Die #Huui	;Save on stack ready for loop
05AF E5	03835	PUSH	Ĥ			;Save I/O redirection bit map
	03836	MOB\$Next\$Device	2:			
05B0 E1	03837	POP	н			Recover redirection bit map
05B1 D1	03838	POP	D			Recover device table addresses pointer
05B2 CD6F06 05B5 B7	03839 03840	CALL ORA	A	\$Device\$	laoie	;Get device table in DE ;Check if any device selected
0586 CAEC05	03841	JZ	MOBSEX	i †		, check if any device science
0020 012000	03842					
05B9 C5	03843	PUSH	B	;<- Ye	s : B	;Save device table addresses pointer
05BA E5	03844	PUSH	н			;Save redirection bit map
	03845	;				
	03846 03847	MOB\$Start\$Watch XRA	1009:			;Reset message needed flag
05BB AF 05BC 32A105	03848	STA	MORSNo	ed\$Messa	0 <i>0</i>	, Neset message needed flag
05BF 010807	03849	LXI		Maximum\$:Time delay
05C2 210906	03850	LXI		Not\$Read		;Address to go to
05C5 CD6D08	03851	CALL	Set\$Wa	tchdog		;Start timer
	03852		/			
0500 01110-	03853	MOB\$Wait: ' LDA	MODAN	ed\$Messa		;Check if watchdog timed out
05C8 3AA105 05CB B7	03854 - 03855	ÖRA	MUB\$Ne	eu priessa	96	, check in watchdog times out
05CC C2EE05	03856	JNZ		tput\$Mes	sage	;Yes, output warning message
05CF CD0F06	03857	CALL		Output\$R		;Check if device ready
05D2 CAC805	03858	JZ	MOB\$Wa			;No, wait
	03859	;				.
05D5 F3	03860	DI				;Interrupts off to avoid
0506 010000	03861 03862	LXI	в.о			; involuntary reentrance ;Turn off watchdog
0509 CD6D08	03863	CALL	Set\$Wa	tchdog		; (HL setting is irrelevant)
	03864			_		
05DC 3A9E05	03865	LDA		aracter		;Get data byte
05DF 4F	03866	MOV	C,A			
05E0 CD2608	03867	CALL	Output	\$Data\$By	te	;Output the data byte
05E3 FB	03868	EI	B	s\$Etx\$Pr	at an - 1	;Deal with ETX/ACK protocol
05E4 CD3A06 05E7 C3B005	03869 03870	CALL JMP		s\$Etx\$Pr xt\$Devic		FDEAT WITH EINIMON PROTOCOL
0JE/ C36003	03870	:	noovne	AL PDE VIC	-	
	03872	, MOB\$Ignore\$Exi	t:			;Ignore timeout error
05EA E1	03873	POP	н			;Balance the stack
O5EB D1	03874	POP	D			
L						

	03875	;		
	03876	MOB\$Exit:		
05EC 79	03877	MOV	A,C	;CP/M "convention"
05ED C9	03878	RET		
	03879	;		
	03880	MOB\$Output\$Mes		
05EE 2A9F05	03881	LHLD	MOB\$Busy\$Message	;Display warning message
05F1 CD5305	03882	CALL	Output\$Error\$Mes	age ; on selected console devices
	03883	MOB\$Request\$Ch		
05F4 CD2F05	03884	CALL	Request\$User\$Cho	
	03885			; action character
05F7 FE52	03886	CPI	'R'	;Retry
05F9 CABB05	03887	JZ	MOB\$Start\$Watchd	9 ;Restart watchdog and try again
05FC FE49	03888	CPI	~I *	;Ignore
05FE CAEA05	03889	JZ	MOB\$Ignore\$Exit	
0601 FE41	03890	CPI	´A'	; Abort
0603 CA360E	03891	JZ	System\$Reset	; Give BDOS function O
0606 C3F405	03892	JMP	MOB\$Request\$Choi	e
	03893	;		
	03894	MOB\$Not\$Ready:		Watchdog timer routine will call this
	03895			routine if the device is busy
	03896			for more than approximately 30 seconds
	03897			Note: This is an interrupt service routine
0609 3EFF	03898	MVI	A. OFFH	;Set request to output message
060B 32A105	03899	STA	MOB\$Need\$Message	, ,
060E C9	03900	RET	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	Return to the watchdog routine;
	03901	:		,
	04000	;#		
	04001	; Check	output ready	
	04002	; check	output (eau)	
	04003			e if the specified device is ready
	04003		eive output data.	e in the specified device is ready
	04004			see if the device has been suspended
	04006 04007		otocol reasons and	IT DIR 15 IOW.
	04007	; NOTE.	This would be deep	NOT check if the USART itself is ready.
	04008			
			inis test is done	in the output data byte routine itself.
	04010 04011	;		
			parameters	
	04012 04013	;	NT N N N N N	
		;	DE -> device tab	le
	04014	;		
	04015		arameters	
	04016	;		
	04017	;		lag set) : Device not ready
	04018	7	A = OFFH (Zero-f	lag clear) : Device ready
	04019	;		
	04020	Check\$Output\$R		
060F 210E00	04021	LXI	H,DT\$Status	;Get device status
0612 19	04022	DAD	D	;HL -> status byte
0613 7E	04023	MOV	A, M	;Get status byte
0614 47	04024	MOV	B,A	;Take a copy of the status byte
0615 E601	04025	ANI	DT\$Output\$Suspen	
0617 C23806	04026	JNZ	COR\$Not\$Ready	;Yes, indicate not ready
	04027			
061A 3E04	04028	MVI	A, DT\$Output\$DTR	;Check if DTR must be high to send
061C A0	04029	ANA	B	Mask with device status from table
061D CA3406	04030	JZ	COR\$Ready	;No, device is logically ready
A/AA A/AAA-	04031			
0620 210000	04032	LXI	H,DT\$Status\$Port	;Set up to read device status
0623 19	04033	DAD	D	
0624 7E	04034	MOV	A, M	;Get status port number
0625 322906	04035	STA	COR\$Status\$Port	;Set up instruction below
	04036			
0628 DB	04037	DB	IN	
	04038	COR\$Status\$Por		
0629 00	04039	DB		<pre>< Set up by instruction above</pre>
062A 4F	04040	MOV	C., A	;Save hardware status
	04041			
062B 210400	04042	LXI	H,DT\$DTR\$Ready	;Yes, set up to check chip status
062E 19	04043	DAD	D	; to see if DTR is high
062F 7E	04044	MOV	Α,Μ	;Get DTR high status mask
0630 A1	04045	ANA	C	;Test chip status
0631 CA3806	04046	JZ	COR\$Not\$Ready	;DTR low, indicate not ready
	04047	;		
	04048	COR\$Ready:		

0634 3EFF	04049	MVI	A, OFFH	;Indicate device ready for output
0636 B7	04050	ORA	A	
0637 C9	04051	RET		
	04052	;		
	04053	COR\$Not\$Ready:		;Indicate device not ready for output
0638 AF	04054	XRA	A	
0639 C9	04055	RET		
	04056	;		
	04200	; #		
	04201	;		
	04202		s ETX/ACK protocol	
	04203	; 		
	04204 04205		outine maintains ETX/ACK	a characters have been output
	04205			is output and the device
	04206			
	04207	; put in	to output suspended state	r interrupt control) will
	04208	; ACK ch ; output	be resumed to the device	I Interrupt Controls will
	04210	•	De l'esdiled to the device	•
	04210	; : Entry	parameters	
	04212	; Entry	parameters	
	04212	;	DE -> device table	
	04214	;	DC / DEVICE (able	
	04215		arameters	
	04216	; =	ar ameter 5	
	04217	;	Message count downdated	(and reset if necessary)
	04218			
	04219	Process\$Etx\$Pr	atocal:	
063A 210E00	04220	LXI	H, DT\$Status	;Check if ETX/ACK protocol enabled
063D 19	04221	DAD	D	,
063E 7E	04222	MOV.	A.M	
063F E610	04223	ANI	DT\$Qutput\$Etx	
0641 C8	04224	RZ		;No, so return immediately
0642 211000	04225	LXI	H,DT\$Etx\$Count	;Yes, so downdate count
0645 19	04226	DAD	D	
0646 E5	04227	PUSH	Ĥ	Save address of count for later
0647 4E	04228	MOV	С, М	;Get LS byte
0648 23	04229	INX	н	, oct 20 D) to
0649 46	04230	MOV	B.M	;Get MS byte
064A OB	04231	DCX	B	,
064B 78	04232	MOV	A, B	
064C B1	04233	ORA	C	Check if count now zero
064D C25706	04234	JNZ	PEP\$Save\$Count	:No
0650 211200	04235	LXI		;Yes, reset to message length
0653 19	04236	DAD	D	,,
0654 4E	04237	MOV	с, м	;Get LS byte
0655 23	04238	INX	H	,
0656 46	04239	MOV	B.M	:Get MS byte
	04240	PEP\$Save\$Count		,
0657 E1	04241	POP	н	;Recover address of count
0658 71	04242	MOV	M,C	Save count back in table
0659 23	04243	INX	Н	
065A 70	04244	MOV	М, В	
	04245	;		
065B B7	04246	ORA	A	;Reestablish whether count hit O
065C C0	04247	RNZ		;No, no further processing required
065D 0E03	04248	MVI	C,ETX	;Yes, send ETX to device
065F F3	04249	DI		Avoids involuntary reentrance
0660 CD2608	04250	CALL	Output\$Data\$Byte	
0663 FB	04251	EI		
0664 210E00	04252	LXI	H,DT\$Status	;Flag device as output suspended
0667 19	04253	DAD	D	· ·
0668 F3	04254	DI		Avoid interaction with interrupts
0669 7E	04255	MOV	Α,Μ	;Get status byte
066A F601	04256	ORI	DT\$Output\$Suspend	;Set bit
066C 77	04257	MOV	M, A	;Save back in table
066D FB	04258	EI		
066E C9	04259	RET		
	04260	;		
	04400	;#		
	04401	;		
	04402		device table	
	04403	;	· · · · · · · -	
	04404		outine scans a 16-bit wor	d, and depending on which is the
	04405			prresponding device table address.
	04406	;		

۲

	04407	; Entry	parameters		
	04408	;	, _, _, _, _, _		
	04409	;	HL = Bit map		
	04410		DE -> Table of	device t	able addresses
	04411	;	The fi	rst addre	ss in the list is called
	04412	;			gnificant bit of the bit map is
	04413			o, and so	
	04414	;		-,	
	04415		arameters		
	04416				
	04417	;	BC -> Current	entry in	device table addresses
	04418	;	DE = Selected		
	04419	,	HL = Shifted b		
	04420	1			was found
	04421				w entirely 0000
	04422	<u>.</u>	2010 11 0	at map ne	antiful, cooc
	04423	; Note:	If HL is 0000H o	n input.	then the first entry in the
	04424	: device	table addresses	will he	returned in DE.
	04425	:			
	04426	, Select\$Device\$	Table		
066F 7C	04427	MOV	A,H	·Get mo	st significant byte of bit map
0670 B5	04428	ORA	L		if HL completely 0
0671 C8	04429	RZ	-		indicating no more bits set
0672 70	04430	MOV	A,L		if the LS bit is nonzero
0673 E601	04431	ANI	1	yoneck	IN THE CO DIT IS HOHZERD
0675 C28006	04432	JNZ	SDT\$₿it\$Set	•Vac ~	eturn corresponding address
0678 13	04433	INX	501#01(#50) D		date table pointer
0679 13	04434	INX	p	,no, up	oute tuble pointer
067A CDDB08	04435	CALL	SHLR	• Shif+	HL right one bit
067D C36F06	04436	JMP	Select\$Device\$;Check next bit
	04437	SDT\$Bit\$Set:	Select#Device#	JODIE	, CHECK HEXT DIT
0680 E5	04438	PUSH	н		bifted bit was
0680 25	04438	MOV			hifted bit map
0681 42 0682 4B	04439	MOV	B,D C,E	FIGKE C	opy of table pointer
0682 4B	04440	XCHG	υ,Ε	· UI	addware in table
0683 EB 0684 5E	04441	MOV	E,M	;nL ->	address in table
0685 23	04442	INX	E,n H		
0686 56	04443	MOV	D, M	• DE ->	selected device table
0000 00	04444	nov	99 M		selected device table registers for another
	04446			; set up ; entr	
0687 E1	04446	POP	н		
0687 E1 0688 CDDB08	04447	CALL			r shifted bit map
0688 CDDB08	04448		SHLR B		bit map right one bit DT address table printer to
0688 03	04449	INX	B		DT address table pointer to
0680 3E01	04450	INX MVI	-	; entr	
068D 3E01 068F B7	04451	MV1 ORA	A, 1 A		te that a one bit was found
			H	; and i	registers are set up correctly
0690 C9	04453	RET			
	04454	;			
	04600	;#			
	04601	;			
	04602		put character		
	04603	;			
	04604				t character from the device
	04605			e table ha	anded over as an input
1	04606	; parame	ter.		
	04607	; 			
	04608	Get\$Input\$Char			.
0691 211900	04609	LXI	H,DT\$Character	≇Count	;Check if any characters have
0694 19	04610	DAD	D		; been stored in the buffer
	04611	GIC\$Wait:			
0695 FB	04612	EI			;Ensure that incoming chars, will
	04613				; be detected
0696 7E	04614	MOV	Α,Μ		;Get character count
0697 B7	04615	ORA	Α		
0698 CA9506	04616	JZ	GIC\$Wait		;No characters, so wait
069B 35	04617	DCR	M		;Down date character count for
	04618				; the character about to be
1	04619				; removed from the buffer
069C 211700	04620	LXI	H,DT\$Get\$Offset		;Use the get offset to access
069F CDF007	04621	CALL	Get\$Address\$in\$	\$Buffer	;Returns HL -> character
	04622				; and with get offset updated
06A2 7E	04623	MOV	Α,Μ		;Get the actual data character
06A3 F5	04624	PUSH	PSW		;Save until later
	04625				
06A4 211900	04626	LXI	H,DT\$Character\$	₿Count	;Check downdated count of chars. in
06A7 19	04627	DAD	D		; buffer, checking if input should be
L					

	04920			
0702 11CE02	04921	LXI	D,DT\$2	;Device 2
0705 CD1607	04922	CALL	. Service\$Device	
	04923			
0708 3E20 070A D3D8	04924 04925	MVI OUT	A,IC\$EOI IC\$OCW2\$Port	;Tell the interrupt controller chip
070C D1	04925	POP	ILDUCW2DFort D	; that the interrupt has been serviced
070D C1	04920	POP	B	Restore registers
070E F1	04928	POP	PSW	
070F 2A8422	04929			;Switch back to user's stack
0712 F9	04930	SPHL		,Swriten back to user a stack
0713 E1	04931	POP	н	
0714 FB	04932	EI		Relenable interrupts in the CPU
0715 09	04933	BET		;Resume pre-interrupt processing
	04934	:		Justana pre internapt processing
	05000	; #		
	05001	;		
	05002	; Serv	ice device	
	05003	;		-
	05004		routine performs	the device interrupt servicing,
	05005	; chec	king to see if the	e device described in the specified
	05006			in DE) is actually interrupting,
	05007			character. Depending on which data character
	05008			will either stack it in the input buffer
	05009			it stream if the buffer is nearly full),
	05010		vill suspend or res	sume the output to the device.
	05011	;	· · · · · · · · · · · · · ·	
	05012		y parameters	
	05013	,		
	05014 05015	:	DE -> device t	.duie
	05015	; Service\$Devi	~~·	
0716 210000	05018	LXI	H,DT\$Status\$Pc	ort ;Check if this device is really
0719 19	05018		n	; interrupting
071A 7E	05019	MOV	Ă, M	;Get status port number
071B 321F07	05020	STA	SD\$Status\$Port	
	05020	014		,
071E DB	05022	DB	IN	;Input status
	05023	SD\$Status\$Po		
071F 00	05024	DB	0;< 9	Set up by instruction above
	05025	;		
0720 210300	05026	LXI	H,DT\$Input\$Rea	ady ;Check if status indicates data ready
0723 19	05027	DAD	D	
0724 A6	05028	ANA	м	;Mask with input ready value
0725 C8	05029	RZ		;No, return to interrupt service
	05030			;Check if any errors have occurred
0726 210700	05031	LXI	H,DT\$Detect\$Er	
0729 19	05032	DAD	D	; interrupting
072A 7E	05033	MOV	A, M	;Get status port number
072B 322F07	05034	STA	SD\$Error\$Port	 Store in instruction below
0705 85	05035	DB	IN	· Ipput array status
072E DB	05036			;Input error status
0705 00	05037	SD\$Error\$Por DB		Set up by instruction above
072F 00	05038 05039		, (S	ACT OF DY INSTRUCTION GDOVE
0730 210800	05039	; LXI	H DISDatester	ror\$Value ;Mask with error bit(s)
0733 19	05040	DAD	D	ioreverse ynask with error bitts/
0734 A6	05041	ANA	M	
0735 CA4707	05043	JZ	SD\$No\$Error	;No bit(s) set
0738 210900	05044	LXI	H,DT\$Reset\$Err	
073B 19	05045	DAD	D	
073C 7E	05046	MOV	A, M	;Get reset port number
073D 324607	05047	STA	SD\$Reset\$Error	\$Port ;Store in instruction below
0740 210A00	05048	LXI	H,DT\$Reset\$Err	or\$Value
0743 19	05049	DAD	D	
07 44 7E	05050	MOV	Α,Μ	;Get reset interrupt value
	05051			
07 4 5 D3	05052	DB	OUT	
	05053	SD\$Reset\$Err		
0746 00	05054	DB	0 ;< 9	Set up in instruction above
	05055	COAN- +C		
	05056	SD\$No\$Error:		
0747 210100	05056 05057	LXI	H,DT\$Data\$Port	
074A 19	05056 05057 05058	LXI DAD	H,DT\$Data\$Port D	; be garbled if an error occurred)
	05056 05057	LXI	H,DT\$Data\$Port	

Figure 8-10. (Continued)

١

	05061			
074F DB	05062	DB	IN	;Input data character
	05063	SD\$Data\$Port:		
0750 00	05064	DB	0 ;< Set up by	instruction above
	05065			
0751 47	05066	MOV	B,A	;Take copy of data character above
0752 210E00	05067	LXI	H, DT\$Status	;Check if either XON or ETX protocols
0755 19	05068	DAD	D	; is currently active
0756 7E	05069	MOV	A, M	;Get protocol byte
0757 E618	05070	ANI	DT\$Output\$Xon + DT\$Outp	out\$Etx
0759 CA8107	05071	JZ	SD\$No\$Protocol	;Neither is active
075C E608	05072	ANI	DT\$Output\$Xon	;Check if XON/XOFF is active
075E C26E07	05073	JNZ	SD\$Check\$if\$Xon	;Yes, check if XON char. input
	05074			;No, assume ETX/ACK active
0761 3E06	05075	MVI	A, ACK	;Check if input character is ACK
0763 B8	05076	CMP	В	
0764 C28107	05077	JNZ	SD\$No\$Protocol	;No, process character as data
	05078	SD\$Output\$Desus	pend:	;Yes, device now ready
	05079			; to accept more data, so indicate
	05080			; output to device can resume
	05081			;The noninterrupt driven output
	05082			; routine checks the suspend bit
0767 7E	05083	MOV	A, M	;Get status/protocol byte again
0768 E6FE	05084	ANI		Suspend ;Preserve all bits BUT suspend
076A 77	05085	MOV	M, A	;Save back with suspend = 0
076B C3D907	05086	JMP	SD\$Exit	Exit to interrupt service without
	05087			; saving data character
	05088	;		
	05089	SD\$Check\$if\$Xon	:	;XON/XOFF protocol active, so
	05090			; if XOFF received, suspend output
	05091			; if XON received, resume output
	05092			;The noninterrupt driven output
	05093			; routine checks the suspend bit
076E 3E11	05094	MVI	A, XON	;Check if XON character input
0770 B8	05095	CMP	B	
0771 CA6707	05096	JZ	SD\$Output\$Desuspend	;Yes, enable output to device
0774 3E13	05097	MVI	A, XOFF	;Check if XOFF character input
0776 B8	05098	CMP	B	. No
0777 C28107	05099	JNZ	SD\$No\$Protocol	;No, process character as data ,
	05100	SD\$Output\$Suspe	nu:	;Device needs pause in output of
	05101			
0774 75	05101			; data, so indicate output suspended
077A 7E	05102	MOV	A,M	;Get status/protocol byte again
077B F601	05102 05103	MOV ORI	A,M DT\$Output\$Suspend	;Get status/protocol byte again ;Set suspend bit to 1
077B F601 077D 77	05102 05103 05104	MOV ORI MOV	A,M DT\$Output\$Suspend M,A	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table
077B F601	05102 05103 05104 05105	MOV ORI	A,M DT\$Output\$Suspend	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without
077B F601 077D 77	05102 05103 05104 05105 05106	MOV ORI MOV	A,M DT\$Output\$Suspend M,A	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table
077B F601 077D 77	05102 05103 05104 05105 05106 05107	MQV ORI MQV JMP	A,M DT\$Output\$Suspend M,A	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without
077B F601 077D 77 077E C3D907	05102 05103 05104 05105 05106 05107 05108	MOV ORI MOV JMP ; SD\$No\$Protocol:	A,M DT\$Output\$Suspend M,A SD\$Exit	;Get status/protocol byte again ;Get suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character
077B F601 077D 77 077E C3D907 0781 211800	05102 05103 05104 05105 05106 05107 05108 05109	MQV ORI MQV JMP ; SD\$No\$Protocol: LXI	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space
077B F601 077D 77 077E C3D907 0781 211800 0784 19	05102 05103 05104 05105 05106 05107 05108 05109 05110	MOV ORI MOV JMP ; SD\$No\$Protocol:	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space ; in the input buffer
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111	MQV ORI JMP SD\$No\$Protocol: LXI DAD MQV	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space ; Get length - 1
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111 05112	MQV ORI MQV JMP \$D\$No\$Protocol: LXI DAD MQV INR	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900	05102 05103 05104 05105 05105 05107 05108 05109 05110 05111 05111 05113	MQV ORI JMP ; SD\$No\$Protoco1: LXI DAD MQV INR LXI	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0786 3C 0787 211900 0784 19	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111 05112 05113 05114	MQV ORI JMP SD\$No\$Protocol: LXI DAD MQV INR LXI DAD	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count	;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer
077B F401 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 0788 BE	05102 05103 05104 05105 05106 05108 05109 05110 05111 05112 05113 05114 05115	MQV ORI JMP SD\$No\$Protocol: LXI DAD MQV INR LXI DAD CMP	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 078B BE 078C CAEBO7	05102 05103 05105 05105 05106 05107 05108 05109 05110 05111 05112 05113 05114	MQV ORI MOV JMP SD\$No\$Protocol: LXI DAD MQV INR LXI DAD CMP JZ	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character</pre>
077B F401 077D 77 077E C3D907 0784 19 0785 7E 0786 3C 0787 211900 0788 BE 0786 C5	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111 05112 05113 05114 05115 05116 05117	MOV ORI JMP SD\$No\$Protocol: LXI DAD MOV INR LXI DAD CMP JZ PUSH	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; :Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 078B BE 078C CAEBO7	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111 05111 05112 05113 05114 05115 05116 05117	MQV ORI MOV JMP SD\$No\$Protocol: LXI DAD MQV INR LXI DAD CMP JZ	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 078B BE 078C CAEB07 078F C5 0790 211600	05102 05103 05104 05105 05106 05107 05107 05107 05110 05112 05113 05114 05115 05116 05117 05118	MQV ORI MOV JMP SD\$No\$Protocol: LXI DAD MOV INR LXI DAD CMP JZ PUSH LXI	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer</pre>
077B F401 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 0788 BE 078C CAEB07 078F C5 0790 211600 0793 CDF007	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111 05112 05113 05115 05115 05115 05118 05119 05120	MQV ORI JMP SD\$No\$Protocol: LXI DAD MQV INR LXI DAD CMP JZ PUSH LXI CALL	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Bet current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 0788 BE 078C CAEB07 078F C5 0790 211600 0793 CDF007 0796 C1	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111 05112 05113 05114 05115 05116 05117 05118 05119 05120	MQV ORI MQV JMP ; SD\$No\$Protocol: LXI DAD MQV INR LXI LXI LXI DAD CMP JZ PUSH LXI LZ LXI CALL POP	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HE -> character position ;Recover input character</pre>
077B F401 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 0788 BE 078C CAEB07 078F C5 0790 211600 0793 CDF007	05102 05103 05104 05105 05106 05107 05108 05107 05110 05112 05113 05114 05115 05116 05116 05117 05118 05119 05120	MQV ORI JMP SD\$No\$Protocol: LXI DAD MQV INR LXI DAD CMP JZ PUSH LXI CALL	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input buffer</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 0788 BE 078C CAEB07 078F C5 0790 211600 0793 CDF007 0796 C1	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111 05112 05113 05114 05115 05114 05115 05116 05119 05120 05121 05122	MQV ORI MQV JMP ; SD\$No\$Protocol: LXI DAD MQV INR LXI LXI LXI DAD CMP JZ PUSH LXI LZ LXI CALL POP	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B ;Update	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input ;Save character in input</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 0788 BE 078C CAEB07 078F C5 0790 211600 0793 CDF007 0796 C1	05102 05103 05104 05105 05106 05107 05107 05107 05110 05112 05113 05114 05115 05116 05117 05118 05117 05120 05121 05122 05123	MQV ORI MQV JMP ; SD\$No\$Protocol: LXI DAD MQV INR LXI LXI LXI DAD CMP JZ PUSH LXI LZ LXI CALL POP	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Euffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B ;Update ; buff	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Campute address of character in ; input buffer ;HE -> character position ;Recover input character ;Save character in input save character in input ;Checking if input should</pre>
077B F401 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 078A 19 0788 BE 078C CAEB07 078F C5 0790 211600 0793 CDF007 0796 C1 0797 70	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111 05112 05113 05114 05115 05114 05115 05116 05119 05120 05121 05122	MQV ORI MQV JMP ; SD\$No\$Protocol: LXI DAD MQV INR LXI LXI LXI DAD CMP JZ PUSH LXI LZ LXI CALL POP	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Euffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B ;Update ; buff	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input ;Save character in input</pre>
077B F601 077D 77 077E C3D907 0784 19 0785 7E 0786 3C 0787 211900 078A 19 0788 BE 0786 CAEB07 078F C5 0790 211600 0793 CDF007 0796 C1 0797 70	05102 05103 05104 05105 05106 05107 05108 05107 05110 05111 05112 05114 05115 05116 05117 05117 05117 05121 05120 05121 05123 05124 05125 05126	MOV ORI MOV JMP ; SD\$No\$Protocol: LXI DAD MOV INR LXI DAD CMP JZ PUSH LXI CALL POP MOV	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B ;Update ; buff ; buff	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Save data character ;Save data character ;Save data character ;HL -> character position ;Recover input character ;Save character in input ;Save character in input ;Save character in input ;Save character in input</pre>
077B F401 077D 77 077E C3D907 0784 19 0785 7E 0785 3C 0787 211900 0788 A19 0788 BE 0780 CAEB07 0780 CAEB07 0790 211600 0793 CDF007 0796 C1 0797 70	05102 05103 05104 05105 05106 05107 05107 05107 05110 05112 05113 05114 05115 05114 05115 05116 05117 05120 05121 05122 05123 05124 05125 05124	MQV ORI MOV JMP SD\$No\$Protocol: LXI DAD MOV INR LXI DAD CMP JZ PUSH LXI CALL POP MOV	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Character\$Count B H,D SD\$Put\$Offset Get\$Address\$In\$Buffer B , buff ; bet H,DT\$Character\$Count D	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input unber of characters in input en, checking if input should :emporarily halted</pre>
077B F601 077D 77 077E C3D907 0784 19 0785 7E 0786 3C 0787 211900 0788 BE 0780 CAEB07 0787 C5 0790 211600 0793 CDF007 0796 C1 0797 70 0798 211900 0798 19 0790 34	05102 05103 05104 05105 05106 05107 05108 05109 05110 05111 05112 05113 05115 05114 05115 05116 05117 05118 05119 05120 05121 05123 05124 05125 05126 05127 05128	MOV ORI MOV JMP SD\$No\$Protocol: LXI DAD MOV INR LXI DAD CMP JZ PUSH LXI CALL POP MOV LXI CALL POP MOV	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B ;Update ; buff ; be t H,DT\$Character\$Count D	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input er, checking if input should emporarily halted ;Update character count</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 0788 BE 0780 CAEB07 078F C5 0790 211600 0793 CDF007 0795 C1 0797 70 0798 211900 0798 19 079C 34 079D 7E	05102 05103 05104 05105 05106 05107 05107 05107 05110 05112 05112 05114 05115 05114 05115 05114 05117 05120 05120 05122 05122 05124 05125 05126 05127 05128	MQV ORI MOV JMP SD\$No\$Protocol: LXI DAD MQV INR LXI DAD CMP JZ PUSH LXI CALL POP MQV	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Euffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Euffer\$Full B H,DT\$Character\$Count B M,B ;Update ; buff , be t H,DT\$Character\$Count D M	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input grave character in input er, checking if input should emporarily halted ;Update character count ;Get updated count</pre>
077B F401 077D 77 077E C3D907 0784 19 0785 7E 0785 7E 0785 3C 0787 211900 0784 19 0788 BE 078C CAEB07 0787 CAEB07 0798 211900 0793 CDF007 0796 C1 0797 70 0798 211900 0798 19 079C 34 079D 7E 079E 7211400	05102 05103 05104 05105 05106 05107 05108 05107 05110 05112 05112 05112 05113 05114 05115 05116 05116 05117 05120 05121 05122 05123 05124 05125 05124 05125 05124 05125 05124	MOV ORI MOV JMP SD\$No\$Protocol: LXI DAD MOV INR LXI DAD LXI CALL POP MOV LXI LXI DAD INR MOV LXI	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B ;Update ; buff ; be t H,DT\$Character\$Count D	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space ; in the input buffer ;Get length = 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL => character position ;Recover input character ;Save character in input buffer ;number of characters in input er, checking if input should .emporarily halted ;Update character count ;Check if current count matches</pre>
077B F601 077D 77 077E C3D907 0784 19 0785 7E 0786 3C 0787 211900 0788 5 0787 211900 0788 BE 0780 CAEB07 0787 C5 0790 211600 0793 CDF007 0796 C1 0797 70 0798 211900 0798 19 0790 7E 0790 7E 0790 7E 0790 7E	05102 05103 05104 05105 05106 05107 05108 05107 05110 05111 05112 05113 05115 05114 05115 05116 05115 05116 05120 05121 05123 05124 05125 05124 05127 05128 05127 05128 05127	MOV ORI MOV JMP ; SD\$No\$Protoco1: LXI DAD MOV INR LXI DAD CMP JZ PUSH LXI CALL POP MOV LXI DAD	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Character\$Count B H,DT\$Character\$Count B H,DT\$Character\$Count B JUpdate ; bet H,DT\$Character\$Count D M A,M A,M	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input grave character in input er, checking if input should emporarily halted ;Update character count ;Get updated count</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 0788 19 0788 BE 0780 CAEB07 078F C5 0790 211600 0793 CDF007 0795 C1 0797 70 0798 211900 0798 19 079C 34 079D 7E 079E 211A00 07A1 19 07A2 BE	05102 05103 05104 05105 05106 05107 05107 05107 05110 05112 05113 05114 05115 05116 05117 05120 05122 05123 05124 05125 05124 05125 05126 05127 05128 05127 05128 05129 05130	MQV ORI MOV JMP SD\$No\$Protocol: LXI DAD MQV INR LXI DAD CMP JZ PUSH LXI CALL POP MQV LXI DAD INR LXI CALL POP MQV	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Euffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Character\$Count B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B ;Update ; buff ; be t H,DT\$Character\$Count D M M H,DT\$Character\$Count D M M H,DT\$Character\$Count D	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input ;Get current in input buffer ;en, checking if input should :emporarily halted ;Update character count ;Check if current count matches ; buffer-full threshold</pre>
077B F601 077D 77 077E C3D907 0784 19 0785 7E 0786 3C 0787 211900 0788 5 0787 211900 0788 BE 0780 CAEB07 0787 C5 0790 211600 0793 CDF007 0796 C1 0797 70 0798 211900 0798 19 0790 7E 0790 7E 0790 7E 0790 7E	05102 05103 05104 05105 05106 05107 05108 05109 05110 05112 05112 05112 05112 05113 05116 05116 05121 05120 05121 05122 05123 05124 05125 05124 05125 05124 05125 05128 05127 05128 05127 05128 05127	MOV ORI MOV JMP SD\$No\$Protocol: LXI DAD MOV INR LXI DAD CMP JZ PUSH LXI CALL POP MOV LXI DAD INR MOV LXI DAD	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B ;Update ; buff ; bet H,DT\$Character\$Count D SUPATE D M A,M JD\$Stop\$Input\$Count D	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input buffer enumber of characters in input er, checking if input should emporarily halted ;Update character count ;Get updated count ;Check if current count matches ; buffer-full threshold ;Not at threshold, check if control</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 0788 19 0788 BE 0780 CAEB07 0787 C5 0790 211600 0793 CDF007 0796 C1 0797 70 0798 211900 0798 19 0790 7E 0796 21400 0791 19 0792 211400 0741 19 0742 BE 0743 C2CE07	05102 05103 05104 05105 05106 05107 05107 05107 05110 05112 05112 05114 05115 05114 05115 05114 05117 05120 05121 05122 05122 05122 05122 05122 05122 05122 05122 05122 05122 05122 05122 05122 05123 05124	MQV ORI MOV JMP ; SD\$No\$Protoco1: LXI DAD MQV INR LXI DAD CMP JZ PUSH LXI CALL POP MQV LXI DAD INR MOV LXI DAD INR	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Buffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Character\$Count B M,B ;Update ; buff ; be t H,DT\$Character\$Count D M A,M H,DT\$Character\$Count D M SD\$Check\$Control	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input buffer enumber of characters in input er, checking if input should emporarily halted ;Update character count ;Get updated count ;Check if current count matches ; buffer-full threshold ;Not at threshold, check if control ; character input</pre>
077B F601 077D 77 077E C3D907 0781 211800 0784 19 0785 7E 0786 3C 0787 211900 0788 19 0788 BE 0780 CAEB07 078F C5 0790 211600 0793 CDF007 0795 C1 0797 70 0798 211900 0798 19 079C 34 079D 7E 079E 211A00 07A1 19 07A2 BE	05102 05103 05104 05105 05106 05107 05108 05109 05110 05112 05112 05112 05112 05113 05116 05116 05121 05120 05121 05122 05123 05124 05125 05124 05125 05124 05125 05128 05127 05128 05127 05128 05127	MQV ORI MOV JMP SD\$No\$Protocol: LXI DAD MQV INR LXI DAD CMP JZ PUSH LXI CALL POP MQV LXI DAD INR LXI CALL POP MQV	A,M DT\$Output\$Suspend M,A SD\$Exit H,DT\$Euffer\$Length\$Mask D A,M A H,DT\$Character\$Count D M SD\$Buffer\$Full B H,DT\$Character\$Count B H,DT\$Put\$Offset Get\$Address\$In\$Buffer B M,B ;Update ; buff ; be t H,DT\$Character\$Count D M M H,DT\$Character\$Count D M M H,DT\$Character\$Count D	<pre>;Get status/protocol byte again ;Set suspend bit to 1 ;Save back in device table ;Exit to interrupt service without ; saving the input character ; ;Check if there is still space ; in the input buffer ;Get length - 1 ;Update to actual length ;Get current count of characters ; in buffer ;Check if count = length ;Yes, output bell character ;Save data character ;Compute address of character in ; input buffer ;HL -> character position ;Recover input character ;Save character in input buffer enumber of characters in input er, checking if input should emporarily halted ;Update character count ;Get updated count ;Check if current count matches ; buffer-full threshold ;Not at threshold, check if control</pre>

07AA 7E	05137	MOV	Α,Μ	;Get status/protocol byte
07AB F602	05138	ORI	DT\$Input\$Suspend	;Indicate input is suspended
07AD 77	05139	MOV	M,A	;Save updated status in table
07AE F5	05140	PUSH	PSW	;Save for later use
07AF E640	05141	ANI	DT\$Input\$RTS	;Check if clear to send to be dropped
07B1 CAC307	05142	JZ	SD\$Check\$Input\$Xon	
07B4 210B00	05143	LXI	H, DT\$RTS\$Contro1\$Pc	
07B7 19	05144	DAD	D	nt , les, get control port number
		MOV		
07B8 7E	05145		A, M	.
0789 320207	05146	STA	SD\$Drop\$RTS\$Port	Store in instruction below;
07BC 210C00	05147	LXI	H,DT\$Drop\$RTS\$Value	2
07BF 19	05148	DAD	D	
07C0 7E	05149	MOV	A, M	;Get value needed to drop RTS
	05150			
07C1 D3	05151	DB	OUT	
	05152	SD\$Drop\$RTS\$Poi	rt:	
07C2 00	05153	DB		- Set up in instruction above
1	05154		. , ,	;Drop into input XON test
	05155	SD\$Check\$Input	t Yon	neck if XON/XOFF protocol being used
	05156	SD#CHECK#INPUL		
0700 54				to temporarily suspend input
07C3 F1	05157	- POP	PSW	Recover status/protocol byte
07C4 E680	05158	ANI	DT\$Input\$Xon	;Check if XON bit set
07C6 CACE07	05159	JZ	SD\$Check\$Control	;No, see if control char. input
07C9 0E13	05160	MVI	C,XOFF	;Yes, output XOFF character
07CB CD2608	05161	CALL	Output\$Data\$Byte	;Output data byte
I	05162	;		• • • • • •
	05163	SD\$Check\$Contro	ol: •CH	eck if control character (other than
	05164			CR, LF, or TAB) input, and update
-	05165			count of control characters in buffer
07CE CD0808	05166	CALL	; Check\$Control\$Char	
07D1 CAD907			Check%Control%Unar	Check if control character
	05167	JZ	SD\$Exit	;No,it is not a control character
0704 211000	05168	LXI	H, BT\$Control\$Count	
07D7 19	05169	DAD	D	
0708 34	05170	INR	M	;Update count of control chars.
	05171	;		
	05172	SD\$Exit:	;Re	set hardware interrupt system
07D9 210500	05173	LXI	H,DT\$Reset\$Int\$Port	
07DC 19	05174	DAD	D ~	
07DD 7E	05175	MOV	A, M	;Get reset port number
07DE B7	05176	ORA	A	;Check if port specified
	05177			; (assumes it will always be NZ)
07DF C8	05178	RZ		;Bypass reset if no port specified
07E0 32E907	05179	STA	SD\$Reset\$Int\$Port	Store in instruction below
07E3 210600	05180	LXI		
07E6 19	05180		H,DT\$Reset\$Int\$Valu D	
0768 19		DAD		
07E7 7E	05182	MOV	A, M	;Get reset interrupt value
	05183			
07E8 D3	05184	DB	OUT	
	05185	SD\$Reset\$Int\$Pc	ort:	
07E9 00	05186	DB	0 ;< Set up	in instruction above
07EA C9	05187	RET		Return to interrupt service routine
	05188	:		
	05189	, SD\$Buffer\$Full:		;Input buffer completely full
07EB 0E07	05190	MVI	C.BELL	;Send bell character as desperate
07ED C32608	05190	JMP	Output\$Data\$Byte	; measure. Note JMP return to
0,20 002000	05192	One	outhut sparaspyle	
				; caller will be done by subroutine
	05193	;		
1	05300	;#		
1	05301	7		
	05302		ress in buffer	
	05303	;		
	05304		outine computes the a	ddress of the next character to
	05305		in a device buffer.	
	05306	;		
	05307		arameters	
	05308	;		
	05309		DE -> appro priate d	evice table
	05310	;		device table of either the
1				
	05311	1	Getauriset	or the Put\$Offset
	05312		•	
		; Exit pa	arameters	
	05313			
	05314	;		
	05314 05315		DE unchanged	
	05314	;	DE unchanged HL -> address in ch	aracter buffer
	05314 05315 05316 05317	;		aracter buffer
	05314 05315 05316	;	HL -> address in ch	aracter buffer

Figure 8-10. (Continued)

07F0 19	05319		DAD	D	;HL -> get/put offset in dev. table
07F1 E5	05320		PUSH	Ĥ	Preserve pointer to table
07F2 4E	05321		MOV	C,M	;Get offset value
07F3 0600	05322		MVI	B,0	;Make into word value
	05323			;Update	offset value, resetting to
	05324			; 0 at	end of buffer
07F5 79	05325		MOV	A,C	;Get copy of offset
07F6 3C	05326		INR	A	;Update to next position
07F7 211800	05327		LXI	H,DT\$Buffer\$Length\$Mask	
07FA 19	05328		DAD	D	
07FB A6	05329		ANA	M	;Mask LS bits with length - 1
07FC E1 07FD 77	05330 05331		POP MOV	H M, A	;Recover pointer to offset in table ;Save new value (set to 0 if nec.)
07FE 211400	05332		LXI	H,DT\$Buffer\$Base	;Get base address of input buffer
0801 19	05333		DAD	n	;HL -> address of buffer in table
0802 7E	05334		MOV	Ā.M	;Get LS byte of address
0803 23	05335		INX	н	;HL -> MS byte of address
0804 66	05336		MOV	Н,М	;H = MS byte
0805 6F	05337		MOV	L,A	;L = LS byte
0806 09	05338		DAD	в	;Add on offset to base
0807 C9	05339		RET		
	05340				
	05341	;			
	05400 05401	;#			
	05401	;	Check c	ontrol character	
	05402	:	CHECK C	Untrut character	
1	05404	;	This ro	utine checks the charact	er in A to see if it is a
	05405	;			R, LF, or TAB. The result is
	05406	;	returne	d in the Z-flag.	.,,
	05407	;			
	05408	;	Entry p	arameters	
	05409	;			
	05410	;		A = character to be che	cked
	05411	;			
	05412 05413	;	Exit pa	rameters	
	05413	7		Zaus status if A dass -	ak anakain a nankurl akausakau
	05414	;		or if it is CR,	ot contain a control character
	05416	,		0/ 1/ 1(15 0)(,	
	05417	,		Nonzero if A contains a	control character other than
	05418	;		CR, LF, or TAB.	
	05419	Check\$C	ontrol\$C	har:	
0808 3E1F	05420		MVI	A, ((- 1	;Space is first noncontrol char.
080A B8	05421		CMP	В	
080B DA2408	05422		JC	CCC\$No	Not a control character
080E 3E0D	05423		MVI	A,CR	;Check if carriage return
0810 B8	05424		CMP	B	No. 1
0811 CA2408	05425		JZ	CCC\$No	Not really a control character
0814 3E0A 0816 B8	05426 05427		MVI CMP	A,LF B	;Check if LF
0817 CA2408	05427		JZ	B CCC\$No	;Not really a control character
081A 3E09	05428		MVI	A, TAB	;Check if horizontal tab
081C B8	05430		CMP	B	
081D CA2408	05431		JZ	CCC\$No	;Not really a control character
0820 3E01	05432		ŇVI	A, 1	;Indicate a control character
0822 B7	05433		ORA	A	
0823 C9	05434		RÉT		
	05435	CCC\$No:			;Indicate A does not contain
0824 AF	05436		XRA	A	; a control character
0825 C9	05437		RET		
	05438	;			
	05500	;#			
	05501 05502	;	Output	data byte	
	05503	;	Julput	and the second s	
	05504	;	This is	a simple polled output	routine that outputs a single
	05505	;	charact	er (in register C on ent	ry) to the device specified in
1	05506	;		ice table.	
	05507	;	Prefera	bly, this routine would	have been re-entrant; however
	05508	;	it does	have to store the port	numbers. Therefore, to use it
	05509	;			pts enabled, the instruction
	05510	;	sequenc	e must be:	
	05511	;		DI :Interr	unte off
	05512 05513	;		DI ;Interr CALL Output\$Data\$Byt	upts off
1	00010	,			. •

	05514	;	EI :Int	errupts on
	05515	;	LI	en apes on
	05516		e to do this may cause	involuntary re-entrance.
	05517	;		······
1	05518	; Entry i	parameters	
	05519	;		
	05520	;	C = character to be	output
	05521	;	DE -> device table	
	05522	;		
0826 C5	05523	Output\$Data\$By PUSH	re: B	;Save registers
0828 05	05524	LXI	ь H,DT\$Output\$Ready	;Get output ready status mask
0827 210200 082A 19	05526	DAD	D	yoet output ready status wask
082B 46	05527	MOV		
0820 210000	05528	LXI	H, DT\$Status\$Port	;Get status port number
082F 19	05529	DAD	D	
0830 7E	05530	MOV	A, M	
0831 323508	05531	STA	ODB\$Status\$Port	Store in instruction below;
	05532	ODB\$Wait\$until	FReady:	
0004 00	05533	`	7.51	. Dood chotus
0834 DB	05534 05535	DB ODB\$Status\$Por	IN	;Read status
0835 00	05536	DBastatusaror		in instruction above
	05537	55	. , v oet up	
0836 A0	05538	ANA	В	;Check if ready for output
0837 CA3408	05539	JZ	ODB\$Wait\$until\$Ready	
083A 210100	05540	LXI	H,DT\$Data\$Port	;Get data port
083D 19	05541	DAD	D	
083E 7E	05542	MOV	A, M	<u>.</u>
083F 324408	05543	STA	ODB\$Data\$Port	Store in instruction below
0842 79	05544 05545	MOV	A,C	;Get character to output
0843 D3	05545	DB	ουτ	
0043 03	05547	ODB\$Data\$Port:	661	
0844 00	05548	DB	0 :< Set up	in instruction above
	05549			
0845 C1	05550	POP	в	Restore registers;
0846 C9	05551	RET		
	05552	;		
	05700	;#		
	05701	1		
1	05702 05703	; Input	status routine	
	05704	; input	status roatine	
1	05705		outine returns a value	in the A register indicating whether
	05706			is/are waiting in the input buffer.
	05707	; Some p	roducts, such as Micro	soft BASIC, defeat normal type-ahead
	05708			racters in order to see if an incoming
	05709			received. In order to preserve
	05710			mstances, the input status return
	05711	; can, a	s an option selected t	y the user, return "data waiting" only ■ a Control-S, -Q or -C. This fools
	05712 05713	; if the ; Micros	oft BASIC into allowing	a type-ahead.
	05714	: .	PHOLO INCO BILOWIN	ay ay providence and a second s
	05715		parameters	
	05716	;		
	05717	;	DE -> device table	
	05718	;		
	05719	; Exitp	arameters	
	05720	;		/
	05721 05722	;	A = 000H 11 no chara buffer	cters are waiting in the input
	05723	,	builei	
	05723	;		
	05725	, Get\$Input\$Stat	us:	
0847 210F00	05726	LXI	H,DT\$Status\$2	;Check if fake mode enabled
084A 19	05727	DAD	D	;HL -> status byte in table
084B 7E	05728	MOV	A, M	;Get status byte
084C E601	05729	ANI	DT\$Fake\$Typeahead	;Isolate status bit
084E CA5B08	05730	JZ	GIS\$True\$Status	;Fake mode disabled
	05731 05732		; • Fal	e mode only indicates data
	05732			dy if control chars. in buffer
0851 211000	05734	LXI	H, DT\$Control\$Count	;Check if any control characters
0854 19	05735	DAD	D	; in the input buffer
0855 AF	05736	XRA	A	;Cheap O
		····		

Figure 8-10. (Continued)

0856 B6	05737		ORA	м	;Set flags according to count
0857 C8	05738		RZ		;Return indicating zero
	05739	GIS\$Data			
0858 AF	05740		XRA	A	;Cheap O
0859 3D	05741		DCR	A	;Set A = OFFH and flags NZ
085A C9	05742		RET		;Return to caller
	05743	; CICAT	****	-	
	05744 05745	GIS\$True	≉otatus		status based on any obaviators
	05746				status, based on any characters y in input buffer
085B 2ASDOF	05747		сныр	CB\$Forced\$Input	;Check if any forced input waiting
085E 7E	05748		MOV	A,M	;Get next character of forced input
085F B7	05749		ORA	A	;Check if nonzero
0860 C25808	05750		JNZ	GIS\$Data\$Ready	;Yes, indicate data waiting
	05751				
0863 211900	05752		LXI	H,DT\$Character\$Count	;Check if any characters
0866 19	05753		DAD	D	; in buffer
0867 7E	05754		MOV	A, M	;Get character count
0868 B7 0869 C8	05755 05756		ORA RZ	A	Empty buffer $A = 0.7 - c^{-1}$
086A C35808	05757		JMP	GIS\$Data\$Ready	;Empty buffer, A = 0, Z-set
0000 000000	05758	;		erefre aug	
	05759	;			
	05900	;#			
	05901	;			
	05902	;	Real ti	me clock processing	
	05903	;	.		
	05904				RTC\$Interrupt routine each time
	05905				e tick count is downdated to see
	05906 05907	;	the con	figuration block is upd	ed. If so, the ASCII time in lated.
	05908	;	the con		
	05909	;	With ead	ch tick, the watchdog c	ount is downdated to see if control
	05910	;	must be	"forced" to a previous	ly specified address on return
	05911	;	from the	e RTC interrupt. The wa	tchdog timer can be used to pull
	05912				rwise be an infinite loop, such
	05913		as wait:	ing for the printer to	come ready.
	05914	:			
	05915 05916	-	Set wate	shdog	
	05917	;	Set watt	LINGON	
	05918		This is	a noninterrupt level s	ubroutine that simply sets the
	05919			g count and address	
	05920	;			
	05921	;	Entry pa	arameters	
	05922	;			
	05923	;			icks before watchdog should
	05924 05925	;		"time out"	contuct will be twoneformed when
	05925	;		<pre>HL = address to which watchdog times</pre>	control will be transferred when
	05926			watchuog times	out -
	05928	; Set\$Watcl	hdog:		
086D F3	05929		DI		;Avoid interference from interrupts
086E 22C100	05930		SHLD	RTC\$Watchdog\$Address	;Set address
0871 60	05931		MOV	H,B	
0872 69	05932		MOV	L,C	
0873 22BF00	05933		SHLD	RTC\$Watchdog\$Count	;Set count
0876 FB	05934		EI		
0877 C9	05935		RET		
	05936 05937	,			
	05937	·	,		
	06000	; * . ;	'		
	06002	•		;Control is re	ceived here each time the
	06003			; real time c	
	06004	RTC\$Inte			
0878 F5	06005		PUSH	PSW	;Save other registers
0879 228622	06006		SHLD	PI\$User\$HL	;Switch to local stack
087C 210000	06007			н,о	· Cat usawis stack
087F 39	06008		DAD SHLD	SP BI#User#Stack	;Get user's stack •Save it
0880 228422 0883 31B022	06009 06010		SHLD LXI	PI\$User\$Stack SP,PI\$Stack	;Save it ;Switch to local stack
0883 318022 0886 C5	06010		PUSH	B	yowitch to local stack
0887 D5	06012		PUSH	D	
0007 00	06012	'		-	
0888 21BE00	06014	1	LXI	H,RTC\$Tick\$Count	;Downdate tick count
L	· · ·				

088B 35	06015	DCR	M	
088C C2B008	06015	JNZ	RTC\$Check\$Watch	log ;Is not at 0 yet
0000 020000	06017	0.112		;One second has elapsed so
088F 3ABD00	06018	LDA	RTC\$Ticks\$per\$S	
0892 77	06019	MOV	M.A	• • • • • • • • •
	06020			;Update ASCII real time clock
0893 11A10F	06021	LXI	D,Time\$in\$ASCII	End ;DE -> 1 character after ASCII time
0896 21BD00	06022	LXI	H, Update\$Time\$E	
		TC\$Update\$Dig:	it:	
0899 1B	06024	DCX	D	;Downdate pointer to time in ASCII
089A 2B	06025	DCX	н	;Downdate pointer to control table
089B 7E	06026	MOV	Α,Μ	;Get next control character
089C B7	06027	ORA	A	;Check if end of table and therefore
089D CAB008	06028	JZ	RTC\$Clock\$Updat	
08A0 FA9908	06029	JM	RTC\$Update\$Digi	t ;Skip over ":" in ASCII time
08A3 1A	06030	LDAX	D	;Get next ASCII time digit
08A4 3C	06031	INR	A	;Update it
08A5 12	06032	STAX	D	; and store it back
08A6 BE	06033	CMP	M	;Compare to maximum value
08A7 C2B008	06034	JNZ	RTC\$Clock\$Updat	ed ;No carry needed so update complete
08AA 3E30	06035	MVI	A, '0'	Reset digit to ASCII O
08AC 12	06036	STAX	D	; and store back in ASCII time
08AD C39908	06037	JMP	RTC\$Update\$Digi	t ;Go back for next digit
	06038 ; 06039 R	TC\$Clock\$Upda	ted.	
		TC\$Check\$Watcl		
08B0 2ABF00	06041		RTC\$Watchdog\$Co	unt ;Get current watchdog count
08B3 2B	06042	DCX	H	;Downdate it
08B4 7C	06043	MOV	A, H	Check if it is now OFFFFH
08B5 B7	06044	ORA	A	
0886 FAC808	06045	JM	RTC\$Dog\$Not\$Set	;It must have been 0 beforehand
08B9 B5	06046	ORA	L	;Check if it is now O
08BA C2C808	06047	JNZ	RTC\$Dog\$NZ	;No, it is not out of time
	06048			-
	06049			;Watchdog time elapsed, so "call"
	06050			; appropriate routine
08BD 21C508	06051	LXI	H,RTC\$Watchdog\$	Return ;Set up return address
08C0 E5	06052	PUSH	н	; ready for return
08C1 2AC100	06053	LHLD	RTC\$Watchdog\$Ad	dress ;Transfer control as though by CALL
08C4 E9	06054	PCHL		
		TC\$Watchdog\$R	eturn:	Control will come back here from
	06056			; the user's watchdog routine
08C5 C3CB08	06057	JMP	RTC\$Dog\$Not\$Set	;Behave as though watchdog not active
	06058 06059 R	TC\$Dog\$NZ:		
08C8 228F00	06060	SHLD	RTC\$Watchdog\$Co	unt ;Save downdated count
VOLO ZZBRUU		TC\$Dog\$Not\$Se		; (Leaves count unchanged)
08CB 3E20	06062	MVI	A, IC\$EOI	Reset the interrupt controller chip
OSCD D3D8	06063	OUT	IC\$OCW2\$Port	meset the interrupt controller chip
0000 0000	06064	001	100000201011	
08CF D1	06065	POP	D	Restore registers from local stack;
08D0 C1	06066	POP	В	,
08D1 2A8422	06067	LHLD	PI\$User\$Stack	;Switch back to user's stack
08D4 F9	06068	SPHL		
08D5 2A8622	06069	LHLD	PI\$User\$HL	;Recover user's registers
08D8 F1	06070	POP	PSW	
08D9 FB	06071	EI		Re-enable interrupts
08DA C9	06072	RET		
	06073. ;			
	06200 ;	#		
	06201 ;			
	06202 ;	Shift	HL Right one bit	
	06203 ;			/
		HLR:		
08DB B7	06205	ORA	A	;Clear carry
OBDC 7C	06206	MOV	А,Н	;Get MS byte
OBDD 1F	06207 06208	RAR		Bit 7 set from previous carry
AODE 47		MOV	LI A	;Bit 0 goes into carry ;But shifted MS byte back
08DE 67	06209		H, A	;Put shifted MS byte back ;Get LS byte
08DF 7D 08E0 1F	06210 06211	MOV RAR	A,L	;Get LS byte ;Bit 7 = bit 0 of MS byte
08E0 1F 08E1 6F	06212	MOV	L,A	Put back into result
08E2 C9	06212	RET	-, -	JIGS WER INCO LEBUIS
VULE UT	06214			
	06215 ;			
	06300 ;	*		

Figure 8-10. (Continued)

Octool : High level diskette drivers Octool : These drivers perform the following functions: Octool : SELDSK Select a secrited disk and return the address of the appropriate disk parameter header Octool : SETDME Set the track number for the maxt read or write Octool : SETDME Set the track number for the next read or write Octool : SETDME Set the track to 0 so that the mext read or write Octool : SETDME Set the track to 0 so that the mext read or write Octool : SETDME Set the track to 0 so that the mext read or write Octool : The optimize the importance of the disk drives in this in the importance of the disk drives in this intervent of the importance of the disk drives in this listing. Octool : The optimize the disk drives in this example BIOS. Octool : Cotool : Status do the disk drives in this example BIOS. Octool : Status do the disk drives in this example BIOS. Status do the disk drives in the set drives in the set additional Octool : Status do the disk drives in the disk drives in the set additional Status do the disk drives in the disk drives in the set additional Octoo : Status do the disk drives in the disk drives in the set additional Status do the disk drives in the disk drives i
06303 These drivers perform the following functions: 06304 SELDEK. Splect a specified disk main return the sddress of 06307 SETTEC. Set the teach number for the next read or write 06308 SETTER. Set the teach number for the next read or write 06309 SETTER. Set the teach address for the next read or write 06300 SETTER. Set the teach address for the next read or write 06301 SECTRAN Translet a logical sector number into a physical 06313 In addition, the high level drivers are responsible for making 06314 In addition, the high level drivers are responsible for making 06315 In official blocking code. This blocking/deblocking code. Stips dbocking/deblocking 06316 to CP/M at though they used a 128-byte sector. They do this 06317 by using blocking/deblocking code. This blocking/deblocking 06328 code is described in more detail later in this listing, 06329 just parameter tables 06320 there are too types of disk drives; standard single=tided, 06321 there are too types of disk drives; standard single=tided, 06321 there are too types of disk drives; standard single=tided, 06321 there are too types of disk drives; standard, 06323
0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0300 0.0311 0.0011 0.0311 0.00110 0.001100000000
06305 j SELDSK. Select a specified disk and return the address of 06306 the appropriate disk prameter header write 06307 SETDEC Set the track number for the next read or write 06308 SETDEN Set the MA (read/write) address for the next read or write 06310 SETTEN Set the track number into a physical 06311 Hott Set the track to 0 so that the next read or write 06313 In addition, the high level drivers are responsible for making 06313 to 07/14 at though they used a 126-bite solt. They do this 06313 to 51/4 floopy disktites that use a 512-byte sector appear 06313 to 07/14 at though they used a 126-byte solt. They do this 06313 to 07/14 at though they used a 126-byte solt. They do this 06314 to 07/14 at though they used a 126-byte solt. They do this 06315 to 0622 06326 for advector the disk drives. standard single-sided. 06327 there are two types of disk drives. standard single-sided. 06328 sinple-density 8", and double-sided. double-density 5 1/4" 06329 tainderd 8" disktes do not need to use the block to 06330 the taindard 8" disktes do not need to use the block to 063331 The s
06306 the appropriate disk parameter header 06307 SETTMA Set the DTA frank/write) address for the next read or write 06307 SETTMA Translate a logical sector number into a physical 06311 HOME 06312 De on Track 0 06313 In addition, the high level drivers are responsible for making 06314 In addition, the high level drivers are responsible for making 06315 the SI/A* floopy disktets that use a SI-byte sector appear 06316 to CP/M as though they used a 128-byte sector. They do this 06317 by using blocking/deblocking code. This blocking/deblocking 06317 by using blocking/deblocking code. This blocking/deblocking 06321 just prior to the code itself. 06322 Just prior to the code itself. 06323 Disk parameter tables 06324 Disk parameter tables 06325 As discussed in Chapter 3, these describe the physical 06326 Characteristics of the disk drivers. In this example BIOS, 06327 there are two types of disk the drivers and angle=sidd. 06328 the standard 8" disktes do not need to use the blocking/ 06331 The standard 8" disktes do not need to use the blocking.
06307 SETTRK Set the track number for the next read or write 06309 SETDMA Set the DMA (read/write) address for the next read or write 06311 SETDMA Set the Lock (a) sector number into a physical 06312 De on Track 0 06314 In addition, the high level drivers are responsible for making 06314 In addition, the high level drivers are responsible for making 06314 In addition, the high level drivers are responsible for making 06315 to off a though they used al 20-byte sector. They do this 06316 to drivers are responsible for making 06317 by using blocking/deblocking code. This blocking/deblocking 06319 just prior to the code itself. 06321 Gasza ; 06322 ; Gasza ; 06323 ; Disk parameter tables 06323 ; discussed in Chapter 3, these describe the physical 06323 ; discussed in the sit drivers, istandard single-sided, 06324 ; As discussed in Chapter 3, these describe the physical 06325 ; single-density P*, and double-density 5 1/4" 06326 ; single-density P*, and double-density 5 1/4" 06332 ; deblocking code, but the 5 1/4" drives do. Therefore an additional </td
06308 j SETSEC Set the sector number for the next read or write 06300 SETTRAN Translate a logical sector number into a physical 06310 SETTRAN Translate a logical sector number into a physical 06311 In addition, the high level of 0 so that the next read or write will 06313 In addition, the high level drivers are responsible for making 06315 the S1/4* floppy disketts that use a S12-byte sector appear 06317 to S17* floppy disketts that use a S12-byte sector appear 06318 to code is described in more detail later in this listing, 06320 code is described in more detail later in this listing, 06323 code is described the describe the physical 06323 code and sector tables 06324 code and sector tables 06325 code and sector tables 06326 timple-density 0", and double-sided, double-density 5.4* 06327 anin-density 0", and double-sided, double-density 5.4* 06328 timple-density 0", and double-sided, double-density 5.4* 06339 to standard 0" diskettes do not need to use the blocking. 06330 to standard 0" diskettes con logical disk's physical 06330 toblocking, code, but the 5.1/4" dives do. Therehore addi
06309 i SETDMA Set the DMA (read/write) address for the next read or write 06311 SETDMA Translate a logical sector number into a physical 06311 Get the track to 0 so that the next read or write will 06313 De on Track 0 06314 In addition, the high level drivers are responsible for making 06315 the 51/4* floopy diskettes that use a 512-byte sector appear 06316 to CF/M as though they used a 128-byte sector. They do this 06317 by using blocking/deblocking code. This blocking/deblocking 06321 just prior to the code issist. 06323 to the code issist. 06323 to bik parameter tables 06324 to the code issist. 06325 to the code issist. 06326 taingle-density B*, and double-sided, double-density 5 1/4* 06327 there are two types of disk drives. In this example blocking. 06328 timple-density B*, and double-sided, double-density 5 1/4* 06329 the standard B* disktes do not need to use the blocking. 06333 byte has been prefixed onto the disk parameter block to 06334 the disk driver, and disk parameter block to 06335 the standard B* diskter or not it meeds d
06310 FECTRAN Translate a logical sector number into a physical 06311 HOME Set the track to 0 to that the next read or write will 06312 De on Track 0 06313 In addition, the high lavel drivers are responsible for making 06314 to E J/A at hough they used a 128-byte sector. They do this 06317 by using blocking/deblocking code. This blocking/deblocking 06317 by using blocking/deblocking code. This blocking/deblocking 06320 just prior to the code itself. 06321 Odsato 06322 Disk parameter tables 06323 characteristics of the disk drives. In this example BLOS, 06323 characteristics of the disk drives is tandard single-density 5 1/4" 06323 single-density P, and double-density 5 1/4" 06323 single-density P, and double-density 5 1/4" 06323 single-density P, and double-density 5 1/4" 06323 these base prefixed onto the disk parameter block ing/ 06323 these consist of disk driver, and disk parameter block ing. 06323 these consist of disk parameter block ing. 06334 perical disk driver, and disk parameter block ing. 06335 these consist of disk parameter
06311 i HOME Set the track to 0 so that the next read or write will 06313 i De on Track 0 06313 i The addition, the high lavel drivers are responsible for making 06315 i to CF/M as though they used a 128-byte sector. They do this 06316 i to CF/M as though they used a 128-byte sector. They do this 06317 by using blocking/deblocking code. This blocking/deblocking 06318 ; code is described in more detail later in this listing, 06321 ; just prior to the code itself. 06323 ; just prior to the code itself. 06324 ; characteristics of the disk drives, in this example BIOS, 06325 ; characteristics of the disk drives, in this example BIOS, 06326 ; characteristics of the disk drives, in this example BIOS, 06327 ; mini-diskettes. 06328 ; single-density 8", and double-sided, double-density 5 1/4" 06329 ; diskette do not need to use the blocking. 06339 ; The standard 8" diskettes do not need to use the blocking. 06339 ; file disk driver, and disk parameter bloc
00312 j be on Track 0 00314 In addition, the high level drivers are responsible for making 00314 the chigh level drivers are responsible for making 00315 the chigh level drivers are responsible for making 00317 by using blocking/deblocking code. This blocking/deblocking 00318 code is described in more detail later in this listing, 00322 just prior to the code itself. 00323 to the code is describe the physical 00324 characteristics of the disk drives. In this semple BIOS, 00325 the standard BP diskettes do not need to use the blocking/ 00326 the standard BP diskettes do not need to use the blocking/ 00327 disk for use, but the 51/4* drives do. Therefore an additional 00328 the standard BP diskettes do not need to use the blocking/ 00331 The standard BP diskettes do not help ad disk's physical 003323 disk driver, and disk parameter block dis. 00332 the standard BP diskettes do not help ad disk's deblocking. 00332 disk driver, and disk parameter block dis. 00333 disk driver, and disk parameter block dis. 00333 tisk driver, and disk parameter block use the blocking.
06313 in addition, the high level drivers are responsible for making 06315 it he 5 1/4" floppy diskettes that use a 512-byte sector appear 06315 it of D/H as though they uses a 128-byte sector. They do this 06316 by using blocking/deblocking code. This blocking/deblocking 06317 just prior to the code itself. 06320 just prior to the code itself. 06321 Ossa 06322 file 06323 as discussed in Chapter 3, these describe the physical 06323 characteristics of the disk drives. In this example BIOS, 06325 characteristics of the disk drives istandard single-sided, 06326 characteristics of the disk drives istandard single-sided, 06327 there are two types of disk drives istandard single-sided, 06328 single-density 0", and double-sided, double-density 5 1/4" 06332 deblocking code, but the 5 1/4" drives do. Therefore an additional 06333 byte has been prefixed on to the disk prameter block to 06334 tell the disk driver, and disk parameter block up 06335 deblocking code, but the 5 1/4" drives do. Therefore an additional 06335 tell the disk driver, and disk parameter block up 06335
06314 In addition, the high level drivers are responsible for making 06315 to CP/M as though they used a 128-byte sector. They do this 06316 to CP/M as though they used a 128-byte sector. They do this 06317 by using blocking/deblocking code. This blocking/deblocking 06321 code is described in more detail later in this listing, 06322 just prior to the code lise! 06323 Disk parameter tables 06324 As discussed in Chapter 3, these describe the physical 06325 thera are two types of disk drives; shandard single-ided, 06326 single-density 8", and double-sided, double-density 5 1/4" 06327 the standard 8" diskettes do not need to use the blocking/ 06328 get in the standard 8" diskettes do not need to use the blocking/ 06337 the standard 8" diskettes do not need to use the blocking/ 06338 the standard 8" diskettes do not need to use the blocking. 06338 the standard 8" diskettes do not need to use the blocking. 06337 the standard 8" diskettes do not need to use the blocking. 06338 the standard 8" diskettes do not need to use the blocking. 06337 the standard 8" diskettes do not need to use the blocking. 06338 the
06315 i the 5 1/4" floppy diskettes that use a 512-byte sector appear 06316 to C/M as though they used a 128-byte sector. They do this 06317 by using blocking/deblocking code. This blocking/deblocking 06320 just prior to the code itself. 06321 is described in more detail later in this listing. 06322 is 06323 Disk parameter tables 06324 As discursed in Chapter 3. these describe the physical 06325 characteristics of the disk drives. In this example BIOS. 06326 simil-diskettes. 06327 there are two types of disk drives standard single-sided. 06328 simil-diskettes do not need to use the blocking/ 06332 single-density 9". and double-sided, double-density 5 1/4" 06332 tell the disk drivers what each logical disk's physical 06333 byte has been prefixed on the disk parameter block to 06334 tell the disk driver, and disk parameter blocks uith 06335 diskette type is, and whether or not it needs deblocking. 06336 tell the disk driver, and disk parameter blocks uith 06336 tell the one parameter block per logical disk. 06337 peseconsist of disk paramet
06316 ; to CP/M as though they used a 128-byte sector. They do this 06317 ; by using blocking code. This blocking/deblocking 06320 ; 06320 ; 06321 ; 06322 ; 06322 ; 06322 ; 06323 ; Disk parameter tables 06324 ; 06325 ; As discussed in Chapter 3, these describe the physical 06325 ; there are two year of disk drives; in andard single-Disk, 06326 ; there are two year of disk drives; in andard single-Disk, 06327 ; there are two year of disk drives; in andard single-Disk, 06328 ; single-density 8", and double-sided, double-density 5 1/4" 06329 ; min-diskettes. 06330 ; 06332 ; the standard 8" diskettes do not need to use the blocking/ 06332 ; deblocking code, but the 5 1/4" drives do. Threfore an additional 06333 ; to the standard 8" diskettes do not need to use the blocking/ 06335 ; deblocking code, but the 5 1/4" drives do. Threfore an additional 06336 ; deblocking code, but the 5 1/4" drives do. Threfore an additional 06337 ; deblocking code, but the 5 1/4" drives do. Threfore an additional 06336 ; diskette type is, and whether or not it needs deblocking. 06337 ; 06337 ; 06338 ; Disk definition tables 06339 ; Disk definition tables 06344 ; 06344 ; 06344 ; 06344 ; 06344 ; 06344 ; 06400 ;# 06404 ; 06404 ; 06404 ; 06404 ; 06404 ; 06405 Disk#Parameter#Headers: plescribed in Chapter 3 06406 ; 06406 ; 06406 Disk Floopy#StSkewtable ;S1/4" diskette) 06407 DW Floopy#StSkewtable ;S1/4" diskette) 06408 DDW Floopy#StSkewtable ;Shares same buffer as A: 06410 ; 06410 ; 06411 ; 06411 ; 06410 ; 06411 ; 06411 ; 06411 ; 06410 ; 06411 ; 06411 ; 06411 ; 06411 ; 06411 ; 06411 ; 06412 ; 06413 DW Floopy#StSkewtable ;Shares same buffer as A: 06450 DW Disk#Stanter#Block ; 06450 DW Disk#Stanter#Block ;
06317 by using blocking/deblocking code. This blocking/deblocking 06319 just prior to the code itself. 06322 just prior to the code itself. 06323 its prior to the code itself. 06324 just prior to the code itself. 06325 just prior to the code itself. 06326 its prior to the code itself. 06327 there are tuo types of disk drives. In this example BIOS. 06328 single-demsity Br, and double-density 5 1/4" 06331 the standard B" diskettes do not need to use the blocking/ 06332 deblocking code, but the 5 1/4" drives do. Therefore an additional 06333 byte has been prefixed onto the disk parameter block ing. 06333 diskette type is, and whether or not it needs deblocking. 06335 diskette type is, and whether or not it needs deblock with 06336 the disk drivers what each logical disk, or the same 06337 there prove block for several logical disk, or the same 06338 Disk definition tables 06339 the disk drivers what each logical disk, or the same 06340 parameter block for several logical disk, or the same 06341 parameter block for several logical disks.
06318 ist for its described in more detail later in this listing, 06310 just prior to the code itself, 06320 ist prior to the code itself, 06321 ist prior to the code itself, 06322 ist prior to the code itself, 06323 ist prior to the code itself, 06325 ist prior to the code itself, 06326 characteristics of the disk drives, standard single-sided, 06327 there are two types of disk drives, standard single-sided, 06328 single-density B*, and double-sided, double-density 5 1/4" 06331 The standard 8* diskettes do not need to use the blocking/ 06333 deblocking code, but the 5 1/4" drives do. Therbore an additional 06334 bolocking code, but the 5 1/4" drives do. Therbore an additional 06335 diskette type is, and whether or not it needs deblocking. 06336 iskette type is, and whether or not it needs deblocking. 06337 Disk definition tables 06338 perior isk driver, and disk parameter blocks with 06339 perior isketfer 06334 perior isketfer 06335 disk driver, and disk parameter blocks with 06336 perior isketfer
06319 just prior to the code itself. 06321 06321 06322 Disk parameter tables 06323 As discursed in Chapter 3, these describe the physical 06324 As discursed in Chapter 3, these describe the physical 06325 As discursed in Chapter 3, these describe the physical 06326 As discursed in Chapter 3, these describe the physical 06327 there are two types of disk drives, is and/ard single-sided, 06338 single-density Sr, and double-sided, double-density 5 1/4" 06337 the standard Sr disk tes do not need to use the blocking/ 06338 belocking code, but the 5 1/4" drives do. Therefore an additional 06339 tell the disk drivers what each logical disk sprameter block to 06334 tell the disk driver, and disk parameter blocks of 06335 diskette type is, and whether or not it needs deblocking. 06336 tell the one parameter block per logical disk, or the same 06337 there consist of disk parameter blocks with 06338 parameter block for several logical disk. 06341 per logical disk driver, and disk parameter block with 06403 iLogical disk Ai (5 1/4" diskette) 06404 iLogical disk Ai (
06320 ; 06322 ; 06322 ; 06322 ; 06322 ; 06322 ; 06322 ; 06322 ; 06324 ; 06325 ; 06326 ; 06327 ; 06328 ; 06329 ; 06329 ; 06320 ; 06321 ; 06322 ; 06333 ; 06331 ; 06332 ; 06333 ; 06333 ; 06334 ; 06335 ; 06336 ; 06337 ; 06338 ; 06339 ; 06339 ; 06339 ; 06339 ; 06339 ; 06339 ; 06339 ; 06330 <td< td=""></td<>
06321 ; 06323 ; Disk parameter tables 06324 ; 06325 ; As discussed in Chapter 3, these describe the physical 06325 ; Characteristics of the disk drives. In this example BIOS, 06326 ; Characteristics of the disk drives. In this example BIOS, 06327 ; There are two types of disk drives, standard single-sided, 06328 ; iheracteristics of the disk drives, standard single-sided, 06331 ; The standard 6" diskettes do not need to use the blocking/ 06332 ; deblocking code, but the 5 1/4" drives do. Therefore an additional 06333 ; byte has been prefixed onto the disk parameter block to, 06334 ; tell the disk driver, and disk parameter block to, 06335 ; Disk definition tables 06336 ; per logical disk driver, and disk parameter block so the same 06341 ; per logical disk driver, and disk parameter block so the same 06343 ; parameter block for several logical disk. 06344 ; iLogical disk A: (5 1/4" diskette) 06400 ; iLogical disk
06322 ; Disk parameter tables 06324 ; As discussed in Chapter 3, these describe the physical 06325 ; As discussed in Chapter 3, these describe the physical 06326 ; characteristics of the disk drives. In this example BIOS, 06327 ; there are two types of disk drives standard single-sided, 06328 ; single-density 8', and double-sided, double-density 5'/4'' 06329 ; single-density 8'', and double-sided, double-density 5'/4'' 06321 ; The standard 8'' diskettes do not need to use the blocking/ 06323 ; byte has been prefixed onto the disk parameter block to 06333 ; byte has been prefixed onto the disk parameter block to 06333 ; byte has been prefixed onto the disk parameter block to 06333 ; Disk definition tables 06334 ; per logical disk driver, and disk parameter block suth 06335 ; Disk definition tables 06336 ; per logical disk driver, and disk parameter block suth 06337 ; per logical disk driver, and disk parameter block suth 06338 ; per logical disk A:
06323Disk parameter tables063250632506325As discussed in Chapter 3, these describe the physical063260632706327there are two types of disk drives. In this example BIOS,06328single-density B", and double-sided, double-density 5 1/4"06329mini-diskettes.06330in the standard B" diskettes do not need to use the blocking/06331The standard B" diskettes do not need to use the blocking/06332deblocking code, but the 5 1/4" drives do. Therefore an additional06333byts has been prefixed onto the disk parameter block to06334byts has been prefixed onto the disk parameter block to06335diskette type is, and whether or not it needs deblocking.06336diskette type is, and whether or not it needs deblocking.06337diskette type is, and whether or not it needs deblock with06338is befinition tables06340it These consist of disk parameter headers, with one entry06341per logical disk driver, and disk parameter blocks with06342it either one parameter block per logical disk.06400it disk drivers06444it disk drivers06445it disk drivers06450Disk Parameter#Headers:06461it disk drivers06462Disk Parameter#Headers:06463it disk drivers06464it disk drivers06464disk drivers06465Disk Floopy#SSReatable06466Disk Floopy#SSReatable06571it
06324 ; 06325 ; As discussed in Chapter 3, these describe the physical 06326 ; Characteristics of the disk drives, is thadard BIOS, 06327 ; there are too types of disk drives, is thadard single=sided, 06328 ; single-density B ⁺ , and double-sided, double-density 5 1/4" 06329 ; min-diskettes. 06330 ; 06333 ; The standard B ⁺ diskettes do not need to use the blocking/ 06333 ; byte has been prefixed onto the disk parameter block to 06333 ; byte has been prefixed onto the disk parameter block to 06333 ; diskette type is, and whether or not it needs deblocking. 06337 ; 06338 ; Disk definition tables 06339 ; 06340 ; These consist of disk parameter headers, with one entry 06341 ; per logical disk driver, and disk parameter blocks with 06342 ; either one parameter block per logical disk. 06404 ; parameter block for several logical disk. 06403 ; 06404 ; logical disk A: (5 1/4" diskette) 06404 ; logical disk A: (5 1/4" skew table 06405 Disk#Parameter#Headers: ; plescribed in Chapter 3 06404 ; logical disk A: (5 1/4" diskette) 06405 DW Flopp#\$\$\$Revetable ;5 1/4" skew table 06406 ; 06406 DW Flopp#\$\$\$Revetable ;5 1/4" diskette) 06407 DW Director#\$Buffer 06412 ; logical disk B: (5 1/4" diskette) 06412 ; logical disk B: (5 1/4" diskette) 06413 DW Flopp#\$\$\$Revetable ;5 1/4" diskette) 06414 ; logical disk B: (5 1/4" diskette) 06415 DW Director#\$Buffer ;5 ame DPB as A: 06416 DW Flopp#\$\$\$Parameter#Block ; fiame DPB as A: 06417 DW Disk\$\$\$Marameter#Block ; fiame DPP as A: 06418 DW Flopp#\$\$\$Parameter#Block ;5 ame DPB as A: 06419 ; 06420 O6415 DW Flopp#\$\$\$Parameter#Block ;5 ame DPF as A: 06419 ; 06420 O6414 DW Disk\$\$\$Marameter#Block ;5 ame DPF as A: 06410 ; 06420 O6415 DW Flopp#\$\$\$Parameter#Block ;5 ame DPF as A: 06420 ; 06420 O6420 DW Flopp#\$\$\$Parameter#Block ;5 ame buffer as A: 06420 ; 06420 O6420 DW Flopp#\$\$\$Parameter#Block ;Private allocation vector 06427 ; 06428
03225 As discussed in Chapter 3, these describe the physical 04327 there are two types of disk drives, in this example BIOS, 04327 there are two types of disk drives, in this example BIOS, 04329 mini-diskettes. 04329 mini-diskettes. 04331 The standard 8" diskettes do not need to use the blocking/ 04332 deblocking code, but the 51/4" drives do. Therefore an additional 04333 byte has been prefixed onto the disk parameter block to 04333 byte has been prefixed onto the disk parameter block to 04333 byte has been prefixed onto the disk parameter block to 04335 diskette type is, and whether or not it needs deblocking. 04337 Gos38 j 04338 j Disk definition tables 04339 j These consist of disk parameter headers, with one entry 04341 j per logical disk hiver, and disks. 04343 j parameter block for several logical disk. 04343 j parameter block for several logical disk. 04343 j lbgsepytSskewalle j 04344 j lbgscal disk hiver, and disks. 04403 j
06326 r characteristics of the disk drives, standard single=sided, 06327 single=density B*, and double=sided, double=density 5 1/4" 06330 nin=diskettes. 06331 The standard B" diskettes do not need to use the blocking/ 06332 deblocking code, but the 5 1/4" drives do. Therefore an additional 06333 deblocking code, but the 5 1/4" drives do. Therefore an additional 06334 tell the disk drivers what each logical disk's physical 06335 diskette type is, and whether or not it needs deblocking. 06336 iskette type is, and whether or not it needs deblocking. 06336 iskette type is, and whether or not it needs deblocking. 06336 iskette type is, and whether or not it needs deblocking. 06336 iskette type is, and whether or not it needs deblocking. 06337 iskette type is, and whether or not it needs deblocking. 06336 iskette type is, and whether or not it needs deblocking. 06337 iskette type is, and whether or not it needs deblocking. 06338 iskette type is, and whether or not it needs deblocking. 06334 iskette type is and the comparater block per logical disk. 06341 iskette block for several logical disk. 06404
06327ithere are two types of disk drives; standard single-sided, 0632906329single-density B', and double-sided, double-density 5 1/4" 0633106331i06332i06333i06333i06333i06333i06333i06333i06333i06333i06334i06335i06336i06337i06337i06338i06337i06337i06338i06337i06337i06338i06337i06337i06337i06337i06337i06337i06337i06337i06337i06337i06337i06331i06331i06332i06333i06333i06341iperiodi06342iii06343iperiodi06404i06405iperiodi06406iperiodi06407DW06408i06409DW06409DW06410i06410i<
06328;single-density B", and double-sided, double-density 5 1/4"06330;mini-diskettes.06330;The standard 8" diskettes do not need to use the blocking/06331;deblocking code, but the 5 1/4" drives do. Therefore an additional06332;deblocking code, but the 5 1/4" drives do. Therefore an additional06333;byte has been prefixed onto the disk parameter block to06334;tell the disk drivers what each logical disk's physical06335;diskette type is, and whether or not in needs deblocking.06336;.06337;Disk definition tables06338;Disk definition tables06331;parameter block for several logical disk.06342;either one parameter block per logical disk.06343;parameter block for several logical disk.06404;parameter block for several logical disk.06404;;Logical disk A: (5 1/4" diskette)068E3 AEO906405DWFloppy55Neutable ;5 1/4" skeu table068E3 0000000006466DWGlopy55Neutable ;5 1/4" diskette)08E1 02206407DWFloppy55Parameter*Block08E206408DWFloppy55Neutable ;Shares same skeu table as A:08E3 000000000646DWFloppy55Neutable ;Shares same skeu table as A:08E1 02206413DWFloppy55Neutable ;Shares same skeu table as A:08E3 0000000006414DWDWprectory8Uffer08E3 0000000006414
06329i06331i06331i06331i06331i06331i06333i06333i06333i06334i06335i06335i06336i06337i06337i06337i06338i06337i06337i06338i06337i06337i06338i06340j06341j06341j06343j06343j06404j06404j06405i06401j06401j06402j#06400j#06400j#06400j#06400j#06400j#06400j#06400j#06400j#06401j06405DW06406DW081506406081906401081106401081906406081906407081106401081106401081206406081406401081506406081906406081906407081106401081106401081106402 <t< td=""></t<>
06330063310633206332063320633306333101106333063341011063351011063361011063370633706338101106338101106339101106330101106331101106332101106333101106334101106336101106337101106338101106339101106340101106340101106340101106341101106342101106343101106344111111110640311110640411
06331iThe standard B" diskettes do not need to use the blocking/06332ideblocking code, but the 5 1/4" drives do. Therefore an additional06333ibyte has been prefixed onto the disk parameter block to06335idiskette type is, and whether or not it needs deblocking.06336idiskette type is, and whether or not it needs deblocking.06337idiskette type is, and whether or not it needs deblocking.06336idiskette type is, and whether or not it needs deblocking.06337idiskette type is, and whether or not it needs deblocking.06337idisk driver, and disk parameter blocks with06341iper logical disk driver, and disk parameter blocks with06342ieither one parameter block per logical disk.06343iparameter block for several logical disk.06404iiLogical disk hi (5 1/4" diskette)06403jiLogical disk hi (5 1/4" diskette)06404iiLogical disk hi (5 1/4" diskette)06405DHFloppy\$5\$Parameter\$Block06406DH0,0,006407DHDisk\$4\$Horkarea06411iiLogical disk B: (5 1/4" diskette)06420Disk\$4\$HorkareaiS 1/4" diskette)06431DHFloppy\$5\$Reatmeter\$Block06432DHFloppy\$5\$Reatmeter\$Block06433DHFloppy\$5\$Reatmeter\$Block06434DHDisk\$4\$Horkarea06435DHFloppy\$5\$Reatmeter\$Block064
06332ideblocking code, but the 5 1/4" drives do. Therefore an additional06333;byte has been prefixed onto the disk parameter block to06334;tell the disk drivers what each logical disk's physical06335;diskette type is, and whether or not it needs deblocking.06336;Disk definition tables06337;0633706338;Disk definition tables06339;These consist of disk parameter headers, with one entry06340;These consist of disk driver, and disk parameter blocks with06341;per logical disk driver, and disk parameter blocks with06343;parameter block per logical disk.06343;parameter block for several logical disk.06400;#;06401;;06402Disk#Parameter#Headers:06403;;06404;Logical disk A: (5 1/4" diskette)068EB 802006405DW068EB 802006406DW069EB 802006407DW068EB 802006408DW069FI 102406411069FI 102406412069FI 102406413069FI 002306414069FI 002306415069FI 002306416069FI 002306417069FI 002306417069FI 002306416069FI 002306417069FI 002306416069FI 002306417069FI 002306417
<pre>06333 ; byte has been prefixed onto the disk parameter block to 06335 ; tell the disk drivers what each logical disk's physical 06335 ; 06337 ; 06337 ; 06338 ; Disk definition tables 06339 ; 06341 ; per logical disk driver, and disk parameter blocks with 06341 ; per logical disk driver, and disk parameter blocks with 06343 ; parameter block per logical disk. 06400 ; 06400 ; 06400 ; 06400 ; 06401 ; 06402 Disk*Parameter*Headers: ;Described in Chapter 3 06404 ; 06402 Disk*Parameter*Headers: ;Described in Chapter 3 06404 ; 06403 ; 06404 ; 06404 ; 06405 DW Floppy%SeKeutable ;5 1/4" diskette) 06406 DW 0.0.0.0 ;Reserved for CP/M 08EB 8022 06407 DW Directory%Buffer 06412 ; 06412 ;Logical disk B: (5 1/4" diskette) 08ET 3609 06412 ;Logical disk B: (5 1/4" diskette) 08ET 8023 06409 DW Disk#A#AHorkarea 06411 ; 06412 ;Logical disk B: (5 1/4" diskette) 08ET 3409 06418 DW Floppy%SeKeutable ;S 1/4" diskette) 08ET 8022 06407 DW Disk#A#AHorkarea 06412 ;Logical disk B: (5 1/4" diskette) 08ET 8022 06417 DW Disk#A#AHorkarea 06ET 000000000000414 DW 0.0.0 ;Reserved for CP/M 08ET 8022 06418 DW Floppy%SeFarameter%Block 08ET 9023 06417 DW Disk#A#AHorkarea ;Private work area 0901 2624 06418 DW Floppy%SeFarameter%Block 08ET 8022 06413 DW Floppy%SeFarameter%Block 08ET 9023 06417 DW Disk#A#AHorkarea ;Private work area 0901 2624 06418 DW Disk#S#BHOrkarea ;Private work area 0901 2620 06421 ;Logical disk C: (8" floppy) 0903 F609 06421 DW Floppy%SeFarameter%Block 06419 ; 06420 ;Logical disk C: (8" floppy) 0903 F609 06421 DW Floppy%SeFarameter%Block 0906 F023 06423 DW Disk@S#AHorkarea ;Private work area 0901 2620 06425 DW Disk@S#AHorkarea ;Private work area 0905 F023 06425 DW Disk@CAHOR ;Seme DEF as A: 0906 F023 06425 DW Disk@CAHOR ;Logical disk C: (8" floppy) 0903 F609 06425 DW Disk@CAHOR ;Private allocation vector 06427 ; 06428 ;Logical disk D: (8" floppy)</pre>
<pre>06334 ; tell the disk drivers what each logical disk's physical 06335 ; diskette type is, and whether or not it needs deblocking. 06336 ; 06337 ; 06338 ; Disk definition tables 06340 ; These consist of disk parameter backs with one entry 06341 ; per logical disk driver, and disk parameter blocks with 06342 ; either one parameter block par logical disk, or the same 06343 ; parameter block for several logical disk. 06344 ; 06400 ; 06401 ; 06400 ; 06401 ; 06400 ; 06401 ; 06402 Disk&Parameter%Headers: ;Described in Chapter 3 06404 ; 10000000000000000000000000000 DW Floppy%5%Revtable ;5 1/4" diskette) 08EB 8002 06407 DW Floppy%5%Revtable ;5 1/4" diskette) 08EB 3009 06408 DW Floppy%5%Revtable ;5 1/4" diskette) 08EB 3009 06408 DW Floppy%5%Revtable ;S 1/4" diskette) 08EF 1024 06410 DW Disk&#AAllocation%Vector 06411 ; 0657 AE09 06413 DW Floppy%5%Revtable ;Shares same skew table as A: 10057 0000000000000000000000000000000000</pre>
<pre>06335 ; diskette type is, and whether or not it needs deblocking. 06337 ; 06337 ; 06338 ; Disk definition tables 06339 ; 06340 ; These consist of disk parameter headers, with one entry 06341 ; per logical disk driver, and disk parameter blocks with 06342 ; either one parameter block per logical disk, or the same 06343 ; parameter block for several logical disks. 06400 ; 06400 ; 06401 ; 06403 ; 06403 ; 06404 ; 06404 ; 06404 ; 06404 ; 06405 Disk#Parameter#Headers: ;Described in Chapter 3 06404 ; 06406 DW Floppy\$5\$Revable ;5 1/4" diskette) 06450 000000006406 DW 0.0.0 ; Heserved for CP/M 06450 000000006406 DW 0.0.0 ; 06410 DW Directory\$Duffer 06410 ; 06411 ; 06410 DW Disk#A\$ANlocationWector 06411 ; 06410 DW Disk#A\$AllocationWector 06411 ; 06410 DW Directory\$Duffer ;Shares same skew table as A: 06410 DW Disk#A\$AllocationWector 06411 ; 06412 ;Logical disk B: (5 1/4" diskette) 06413 DW Floppy\$5\$Reameter\$Block ;Same DFB as A: 08FB 0000000006414 DW 0.0.0 ;Reserved for CP/M 06FB 34609 06415 DW Floppy\$5\$Reameter\$Block ;Same DFB as A: 08FB 0020 06416 DW Floppy\$5\$Reameter\$Block ;Same DFB as A: 08FB 3409 06416 DW Floppy\$5\$Reameter\$Block ;Same DFB as A: 08FB 0020 06417 DW Directory\$Duffer ;Shares same skew table as A: 08FB 0020 06418 DW Floppy\$5\$Reameter\$Block ;Same DFB as A: 08FB 03009 06416 DW Floppy\$5\$Reameter\$Block ;Same DFB as A: 08FD 3409 06416 DW Floppy\$5\$Reameter\$Block ;Same DFB as A: 096F 0023 06417 DW Disk\$B\$Allocation\$Vector ;Private allocation vector 06420 ;Logical disk C: (8" floppy) 0903 F609 06421 DW Floppy\$5\$Reameter\$Block 0900 4009 06422 DW Directory\$Buffer ;Shares same buffer as A: 0900 4009 06423 DW Floppy\$5\$Reameter\$Block 0900 F0023 06425 DW Directory\$Buffer ;Private allocation vector 06427 ; 06428 ;Logical disk D: (8" floppy)</pre>
06336 ; 06337 ; 06338 ; Disk definition tables 06338 ; Disk definition tables 06340 ; These consist of disk parameter headers, with one entry 06341 ; per logical disk driver, and disk parameter blocks with 06342 ; either one parameter block per logical disk, or the same 06343 ; parameter block for several logical disks. 06404 ; 06400 ;# 06402 Disk%Parameter%Headers: ;Described in Chapter 3 06404 ; 06403 ; 06405 DW Flopy\$5%Kewtable ;5 1/4" diskette) 08E3 00000000006406 DW Flopy\$5%Kewtable ;5 1/4" skew table 08E5 00000000006406 DW Flopy\$5%Kewtable ;5 1/4" skew table 08E5 00000000006406 DW Flopy\$5%Kewtable ;5 1/4" diskette) 08E5 00000000006406 DW Flopy\$5%Kewtable ;5 Nares same skew table as A: 0651 1024 06410 DW Flopy\$5%Kewtable ;5Nares same skew table as A: 0658 0000000006414 DW 0,0,0 ;Reserved for CP/M 0658 D020 06415 DW Flopp\$5%Kewtable ;Shares same skew table as A: 0659 06000000006414 DW 0,0,0 ;Reserved for CP/M 0658 D020 06415 DW Directory\$Dwffer ;Shares same buffer as A: 0659 060000000006414 DW Jisk\$\$%Allocation\$Vector ;Private work area 0961 262 06415 DW Disk\$\$%Allocation\$Vector ;Private allocation vector 06419 ; 06420 ;Logical disk C: (8" floppy) 0903 F609 06421 DW Flopp\$8%Revtable ;S" skew table 0905 00000000006422 DW 0.0,0 ;Reserved for CP/M 0908 B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 0909 4040 06424 DW Flopp\$8%Revtable ;S" skew table 0900 4040 06424 DW Flopp\$8%Revtable ;S" skew table 0900 5409 06421 DW Flopp\$8%Revtable ;S" skew table 0900 5409 06421 DW Flopp\$8%Revtable ;S" skew table 0900 40409 06423 DW Directory\$Buffer ;Shares same buffer as A: 0900 40409 06424 DW Flopp\$8%Revtable ;Private allocation vector 06427 ; 06428 ;Logical disk D: (8" floppy)
06337 ; 06338 ; Disk definition tables 06339 ; These consist of disk parameter headers, with one entry 06340 ; per logical disk driver, and disk parameter blocks with 06342 ; either one parameter block per logical disk, or the same 06343 ; parameter block for several logical disk, or the same 06344 ; arameter block for several logical disk, or the same 06400 ;#
06338Disk definition tables06339These consist of disk parameter headers, with one entry06340These consist of disk parameter blocks with06341per logical disk driver, and disk parameter blocks with06342i either one parameter block per logical disk, or the same06343parameter block for several logical disks.06344i06400i06401i06402Disk#Parameter#Headers:06403i06404i06404j06405DW06500000000006406DW06500000000006406DW06500000000006406DW0650DW0650DW0650DW0650DW0651DW0651DW0652064070853AE0906408DW0651DW0652064090853AE0906411j0651DW06520641308750000000004140875DW0675000000000414DWDisk#A#Workarea0875000000000414DWDisk#B#Workarea087506417DWDisk#B#Workarea08750000000000414DWDisk#B#Workarea07122406419j10000DE06420DW0585DW06421DW0595<
06330;These consist of disk parameter headers, with one entry 06341 ;06341;per logical disk driver, and disk parameter blocks with 06342 ;06342;either one parameter block per logical disk, or the same 06343 ;06344;06404 ;06400;#06401;06402Disk#Parameter#Headers: 06403 ;06403;06404;06404;0652064050652064050652064070652064070653DW0652064070653DW0654DW065500000000064060865DW065006407085110240651064080851102406411;0653AE0906412;Logical disk B: (5 1/4" diskette)0657064130857000000000064140857D00000000064140857D00000000064140857D0206417DW012624064180857000000000064140857000000000064140857D00000000064140857D00000000064140857D00000000064140857D00000000064140857D00000000064140857D00000000064140857D00000000064140857D00000000064140857D00000000064140857D02
06340:These consist of disk parameter headers, with one entry oper logical disk driver, and disk parameter blocks with 0634206341:per logical disk driver, and disk parameter blocks with obside06342:either one parameter block per logical disk. obside06343:parameter block for several logical disk. obside06344:
06342 ; per logical disk driver, and disk parameter blocks with 06343 ; parameter block per logical disk, or the same 06343 ; parameter block for several logical disk, or the same 06344 ; 06400 ; 06401 ; 06402 Disk\$Parameter\$Headers: ;Described in Chapter 3 06403 ; 06404 ;Logical disk A: (5 1/4" diskette) 08E3 AEO9 06405 DW Floppy\$5\$Skeutable ;5 1/4" skeu table 08E5 00000000000000000000000000000000000
06342:either one parameter block per logical disk, or the same06343;parameter block for several logical disks.06344;06401;06402Disk\$Parameter\$Headers:;Described in Chapter 306403;06404;Logical disk A: (5 1/4" diskette)08E3 AE0906405DW08E5 0000000006406DW0,0,008E5 802206407DW08E5 802206409DW08E5 802306409DW08E7 802306409DW08E7 802306410DW08E7 802306410DW08E7 80290641108E7 8000000000000000000414DW08E7 800000000000414DW08E7 80000000000414DW08E7 800000000000414DW08E7 80000000000414DW08E7 80000000000414DW08E7 80000000000414DW08E7 80000000000414DW08F8 80220641508F1 90230641708F8 80220641609F1 26240641809F0 26417DW0903 F60906421090400064220903 F609064210904000642209040906424090409064240905 8020064240906 4020;Logical disk C: (S" floppy)0908 F602064260907 F603064260907 F023064260907 F023064260901 30206426
06343 ; parameter block for several logical disks. 06304 ; 06400 ; 06402 Disk\$Parameter\$Headers: ;Described in Chapter 3 06403 ; 06404 ;Logical disk A: (5 1/4" diskette) 08E3 AE09 06405 DW 08E5 0000000006406 DW 0,0,0 08E5 0020 06407 DW Directory\$Buffer 08E5 0622 06407 DW 08E5 0023 06408 DW Floppy\$5\$Parameter\$Block 08E7 1024 06410 DW Disk\$A\$Allocation\$Vector 06411 ; :Logical disk B: (5 1/4" diskette) 08F3 AE09 06413 DW Floppy\$5\$Parameter\$Block 08F3 6000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 0022 06415 DW Floppy\$5\$Parameter\$Block ;Same DFB as A: 08F5 0023 06417 DW Disk\$B\$Allocation\$Vector ;Private work area 0961 220 06413 DW Flopp\$S\$Sarameter\$Block ;Same DFB as A:
06344 ; 06400 ; 06401 ; 06402 Disk\$Parameter\$Headers; ;Described in Chapter 3 06403 ; 06404 ;Logical disk A: (5 1/4" diskette) 08E3 AE09 06405 DW 08E5 0000000006406 DW 0,0,0 ;Reserved for CP/M 08E5 0000000006406 DW Disk\$farameter\$Block 08E5 08E5 0000000006406 DW Disk\$fafarameter\$Block 08E5 08E5 00000000006406 DW Disk\$fafallocation\$Vector 08E5 08E5 0000000006406 DW Disk\$fafallocation\$Vector
06400 ;# 06401 ; 06402 Disk\$Parameter\$Headers: ;Described in Chapter 3 06403 ; 06404 ;Logical disk A: (5 1/4" diskette) 08E3 AE09 06405 DW 08E5 000000006406 DW 0,0,0 ;Reserved for CP/M 08EB 8022 06408 DW Floppy\$5\$Parameter\$Block 08E1 3409 06408 DW Floppy\$5\$Parameter\$Block 08E1 1024 06410 DW Disk\$A\$Workarea 08F1 1024 06411 ; ;Logical disk B: (5 1/4" diskette) 08F3 AE09 06413 DW Floppy\$5\$Skewtable ;Shares same skew table as A: 08F5 000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 000000006414 DW Floppy\$5\$Parameter\$Block ;Same DPB as A: 08F1 3409 06412 jLogical disk C: (8" floppy) ;Private work area 0901 2624 06417 DW Disk\$B\$Allocation\$Vector ;Private allocation vector 06420 ;Logical disk C: (8" floppy) 0905 So00000006422
06401
O6402 O6403 O6404 Disk\$Parameter\$Headers: (Logical disk A: (5 1/4" diskette) 08E3 AE09 06404 ;Logical disk A: (5 1/4" diskette) 08E5 000000006406 DW Flopp\$5\$\$ketable ;5 1/4" skew table 08E5 0000000006406 DW 0,0,0 ;Reserved for CP/M 08EB 8022 06407 DW Directory\$Buffer 08E5 8020 06408 DW Flopp\$5\$Parameter\$Block 08E7 8023 06409 DW Disk\$4\$Ablocation\$Vector 06411 ; ;Logical disk B: (5 1/4" diskette) 08F3 AE09 06413 DW Flopp\$5\$\$kewtable ;Shares same skew table as A: 08F5 0000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 0000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 0023 06415 DW Flopp\$5\$Parameter\$Block ;Same DPB as A: 08F7 D230 06416 DW Flopp\$5\$Parameter\$Block ;Private allocation vector 06419 ; ; ;Logical disk C: (8" flopp) ;0905 000000006422 0905 00000000006422 DW Flo
0403 ; ;Logical disk A: (5 1/4" diskette) 0683 AE09 06404 ;Logical disk A: (5 1/4" diskette) 0885 AE09 06405 DW Floppy\$5\$kewtable ;5 1/4" diskette) 0885 0000000006406 DW 0,0,0 ;Reserved for CP/M 0885 000000006406 DW Directory\$Buffer ;Reserved for CP/M 0885 000000006408 DW Floppy\$5\$Parameter\$Block ;Destaf4\$Workarea 0887 1024 06409 DW Disk\$4\$Workarea 0887 3E09 06411 ; ;Logical disk B: (5 1/4" diskette) 0887 3E09 06413 DW Floppy\$5\$kewtable ;Shares same skew table as A: 0887 00000000006414 DW 0,0,0 ;Reserved for CP/M 0887 0000000006414 DW Flopp\$5\$Parameter\$Block ;Same DPB as A: 0887 023 06417 DW Disk\$8\$Horkarea ;Private work area 0901 2624 06418 DW Flopp\$8\$Skewtable ;8" skew table 2067 0 CP/M 0905 0000000006422 DW Flopp\$8\$Skewtable ;8" skew table 2067 0 CP/M 0905 0000000006422 DW Flopp\$8\$Skewtable ;8" skew table 206423 090
06404 jLogical disk A: (5 1/4" diskette) 08E3 AE09 06405 DW Floppy\$5\$Skevtable ;5 1/4" diskette) 08E5 0000000006406 DW 0,0,0 ;Resrved for CP/M 08E5 0022 06407 DW Directory\$Buffer 08E5 0023 06408 DW Floppy\$5\$Parameter\$Block 08EF 8023 06409 DW Disk\$4\$Ablocktorsea 08F1 1024 06410 DW Disk\$4\$Ablocktorsea 06411 ;
08E3 AE09 06405 DW Floppy\$5\$kewtable ;5 1/4" skew table 08E5 00000006406 DW 0,0,0 ;Reserved for CP/M 08EB 8022 06407 DW Directory\$Buffer 08EB 8023 06408 DW Floppy\$5\$Parameter\$Block 08EF 8023 06409 DW Disk\$4\$Ablocation\$Vector 06411 ;
08E5 00000000006406 DW 0,0,0 ;Reserved for CP/M 08EB 8022 06407 DW Directory\$Buffer 08ED 8023 06408 DW Floppy\$5\$Parameter\$Block 08ET 8023 06409 DW Disk\$A\$AUcation\$Vector 08E1 1024 06411 ; ;Logical disk B: (5 1/4" diskette) 08F3 06412 ;Logical disk B: (5 1/4" diskette) ; 08F3 06412 ;Logical disk B: (5 1/4" diskette) 08F3 06412 ;Logical disk B: (5 1/4" diskette) 08F3 06412 ;Logical disk B: (5 1/4" diskette) 08F3 0000000006414 DW Floppy\$5\$Rarameter\$Block ;Same DPB as A: 08F5 0023 06417 DW Disk\$B\$Allocation\$Vector ;Private work area 0901 2624 06418 DW Disk\$B\$Allocation\$Vector ;Private allocation vector 06420 ;Logical disk C: (8" floppy) ;Private work area ;Private allocation vector 0901 26424 DW Floppy\$8\$\$Rewtable ;Bressame buffer as A: 0900 64021 DW
08EB 8022 06407 DW Directory\$Buffer 08ED 3409 06408 DW Floppy\$5\$Parameter\$Block 08EF 8023 06409 DW Disk\$4\$Workarea 08EF 8023 06410 DW Disk\$4\$A\$llocation\$Vector 06411 ; :Logical disk B: (5 1/4" diskette) 08F3 AE09 06413 DW Flopp\$5\$\$ketable ;Shares same skew table as A: 08F3 AE09 06415 DW Directory\$Buffer :Shares same buffer as A: 08F5 0000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 0000000006415 DW Directory\$Buffer :Shares same buffer as A: 08F5 00022 06415 DW Disk\$8\$Morkarea ;Private work area 0901 2624 06418 DW Disk\$8\$Morkarea ;Private work area 0901 2624 06418 DW Disk\$8\$Morkarea ;Private work area 0902 6000000006422 DW Flopp\$8\$Skewtable ;8" skew table 0907 0000000006422 DW Directory\$Buffer ;Shares same buffer as A: 0900 4409 06424 DW Flopp\$8\$Farameter\$Block PO
08ED 3409 06408 DW Floppy\$5\$Parameter\$Block 08EF B023 06409 DW Disk\$4\$Workarea 08F1 1024 06411 ; isk\$4\$Workarea 08F3 AE09 06413 DW Floppy\$5\$Skewtable ;Shares same skew table as A: 08F3 AE09 06413 DW Floppy\$5\$Skewtable ;Shares same buffer as A: 08F5 0000000006414 DW 0,0,0 ;Reserved for CP/M 08FB B022 06416 DW Floppy\$5\$Parameter\$Block ;Same DPB as A: 08FD 3409 06417 DW Disk\$8\$Horkarea ;Private work area 0901 2624 06417 DW Disk\$8\$Horkarea ;Private allocation vector 06419; ;
08EF B023 06409 DW Disk\$4\$Workarea 08F1 1024 06410 DW Disk\$4\$A\$Norkarea 06411 ;
08F1 1024 06410 DW Disk\$4\$Allocation\$Vector 06412 ;Logical disk B: (5 1/4" diskette) 08F3 AE09 06413 DW Flopp\$5\$Skewtable ;Shares same skew table as A: 08F5 0000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 0000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 0022 06415 DW Directory\$Buffer ;Shares same buffer as A: 08FF D023 06417 DW Disk\$B\$Morkarea ;Private work area 0901 2624 06418 DW Disk\$B\$Allocation\$Vector ;Private allocation vector 06420 ;Logical disk C: (8" floppy) 0905 0000000006422 DW Flopp\$88\$Skewtable ;8" skew table 0901 6402 06423 DW Flopp\$88\$Skewtable ;8" skew table 0905 0902 06423 DW Flopp\$88\$Skewtable ;8" skew table 0905 0000000006422 DW Directory\$Buffer ;Shares same buffer as A: 0900
06411 ; ;Logical disk B: (5 1/4" diskette) 08F3 AE09 06412 jLogical disk B: (5 1/4" diskette) 08F3 AE09 06413 DW Floppy\$5\$kewtable ;Shares same skew table as A: 08F5 0000000006414 DW 0,0,0 ;Reserved for CF/M 08FB 0022 06415 DW Directory\$Buffer ;Shares same buffer as A: 08FD 3409 06416 DW Floppy\$5\$Parameter\$Block ;Same DFB as A: 08FF D023 06417 DW Disk\$B\$Morkarea ;Private work area 0901 2624 06418 DW Disk\$B\$Allocation\$Vector ;Private allocation vector 06420 ;Logical disk C: (8" floppy) 0905 Sou00000006422 DW 0,0,0 ;Reserved for CP/M 0905 B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 0905 B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 0905 B022 06424 DW Floppy\$8\$Parameter\$Block 0907 Flopp 06424 DW Floppy\$8\$Parameter\$Block ;Private work area 0911 3C24 06426 DW Disk\$5\$C\$
06412 ;Logical disk B: (5 1/4" diskette) 08F3 AE09 06413 DW Floppy\$5\$\$kevtable ;Shares same skew table as A: 08F5 00000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 0022 06415 DW Directory\$Buffer ;Shares same buffer as A: 08F5 0023 06416 DW Floppy\$5\$Parameter\$Block ;Same DPB as A: 08FF D023 06417 DW Disk\$B\$Workarea ;Private work area 0901 2624 06418 DW Disk\$B\$Allocation\$Vector ;Private allocation vector 06420 ;Logical disk C: (8" floppy) 06420 ;Logical disk C: (8" floppy) 0905 0000000006422 DW Floppy\$8\$Skewtable ;Shares same buffer as A: 0908 B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 0900 4409 06424 DW Floppy\$8\$Parameter\$Block ;Private work area 0901 3C24 06425 DW Disk\$5\$Morkarea ;Private work area 0911 3C24 06426 DW Disk\$5\$Allocation\$Vector ;Private allocation vector
08F3 AE09 06413 DW Floppy\$5\$Skewtable ;Shares same skew table as A: ;Reserved for CP/M 08F5 000000006414 DW 0,0,0 ;Reserved for CP/M 08F5 00000006414 DW Directory\$Buffer ;Shares same buffer as A: 08F5 00000006414 DW Floppy\$5\$Parameter\$Block ;Same DPB as A: 08F5 0023 06416 DW Floppy\$5\$Parameter\$Block ;Same DPB as A: 08F5 0023 06417 DW Disk\$8\$Horkarea ;Private work area 0901 2624 06418 DW Disk\$8\$Hallocation\$Vector ;Private allocation vector 06420 ;Logical disk C: (8" floppy) 0905 \$609 06421 DW Floppy\$8\$Skewtable ;8" skew table 0901 500000000006422 DW Floppy\$8\$Skewtable ;8" skew table 0905 \$000000006422 DW Directory\$Buffer ;8" skew table 0900 8002 06423 DW Floppy\$8\$Skewtable ;8" skew table ;9" skew table 0901 4409 06424 DW Floppy\$8\$Shorkarea ;Private work area 0901 3024 06425 DW Disk\$5\$Morkarea ;Private allocation vector 0911 3C24
08F5 0000000000414 DW 0,0,0 ;Reserved for CP/M 08FB B022 06415 DW Directory\$Buffer ;Shares same buffer as A: 08FD 3409 06416 DW Directory\$Buffer ;Shares met buffer as A: 08FD 3409 06417 DW Disk\$B\$Workarea ;Private work area 0901 2624 06418 DW Disk\$B\$Allocation\$Vector ;Private allocation vector 06419 ;
08FB B022 06415 DW Directory\$Buffer iShares same buffer as A: 08FD 3409 06416 DW Floppy\$5\$Parameter\$Block ;Same DPB as A: 08FD 023 06417 DW Disk\$BWorkarea iPrivate work area 0901 2624 06418 DW Disk\$B\$Allocation\$Vector iPrivate work area 0901 2624 06418 DW Disk\$B\$Allocation\$Vector iPrivate allocation vector 06420 iLogical disk C: (8" floppy) 06420 06421 DW 0905 0000000006422 DW Floppy\$8\$Skewtable is% rskew table 0905 00000000006422 DW Directory\$Buffer iShares same buffer as A: 0908 B022 06423 DW Directory\$Buffer iShares same buffer as A: 0909 F023 06424 DW Floppy\$8\$Parameter\$Block iPrivate work area 0901 3C24 06425 DW Disk\$C\$Allocation\$Vector iPrivate work area 0911 3C24 06426 DW Disk\$C\$Allocation\$Vector iPrivate allocation vector 06427 itogical disk D: (8" floppy) iprivate allocation vector iprivate allocation vector
08FD 3409 06416 DW Floppy\$5\$Parameter\$Block ;Same DPB as A: 08FF D023 06417 DW Disk\$B\$Workarea ;Private work area 0901 2624 06418 DW Disk\$B\$Allocation\$Vector ;Private allocation vector 06419 ;
08FF D023 06417 DW Disk\$B\$Workarea ;Private work area 0901 2624 06418 DW Disk\$B\$Allocation\$Vector ;Private allocation vector 06419 ; Disk\$B\$Allocation\$Vector ;Private allocation vector 06420 ;Logical disk C: (8" floppy) 0903 F609 06421 DW 0905 000000000006422 DW 0,0,0 ;Reserved for CP/M 0908 B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 0900 4409 06424 DW Floppy\$8\$Parametr\$Block ;Private work area 0901 3C24 06425 DW Disk\$C\$Allocation\$Vector ;Private allocation vector 06427 ;
0901 2624 06418 DW Disk\$B\$Allocation\$Vector :Private allocation vector 06420 ;Logical disk C: (8" floppy) 0903 F609 06421 DW Flopp\$8\$Skewtable ;8" skew table 0905 0000000006422 DW Flopp\$8\$Skewtable ;8" skew table 0908 B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 0908 4409 06424 DW Flopp\$8\$Farameter\$Block ;Private work area 0907 3024 06425 DW Disk\$\$C\$Allocation\$Vector ;Private work area 0911 3C24 06426 DW Disk\$\$C\$Allocation\$Vector ;Private allocation vector 06427 ;
06419 ;Logical disk C: (8" floppy) 0903 F609 06421 DW Floppy\$8\$Skewtable ;8" skew table 0905 0000000006422 DW 0,0,0 ;Reserved for CP/M 0908 B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 0900 06424 DW Floppy\$8\$Parameter\$Block 090F F023 06425 DW Disk\$C\$Allocation\$Vector ;Private work area 0911 3C24 06426 DW Disk\$C\$Allocation\$Vector ;Private allocation vector 06427 ;Logical disk D: (8" floppy) j:Logical disk D: (8" floppy) j:Logical disk D: (8" floppy)
06420 ;Logical disk C: (8" floppy) 0903 F609 06421 DW Floppy\$8\$Skewtable ;8" skew table 0905 00000000006422 DW 0,0,0 ;Reserved for CP/M 0908 B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 0908 F023 06424 DW Floppy\$8\$Parametr\$Block ;Private work area 0901 3C24 06426 DW Disk\$C\$Allocation\$Vector ;Private allocation vector 06427 ;
0903 F609 06421 DW Floppy\$8\$Skewtable ;8" skew table 0905 00000000006422 DW 0,0,0 ;Reserved for CP/M 0908 B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 0909 4409 06424 DW Floppy\$8\$Farameter\$Block ;Private work area 0901 3C24 06425 DW Disk\$5\$Morkarea ;Private allocation vector 0911 3C24 06426 DW Disk\$5\$Morkarea ;Private allocation vector 06427 ;
0905 0000000006422 DW 0,0,0 ;Reserved for CP/M 090B B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 090D 4009 06424 DW Floppy\$BeFarameter\$Block 090F F023 06425 DW Disk\$C\$Workarea ;Private work area 0911 3C24 06426 DW Disk\$C\$Allocation\$Vector ;Private allocation vector 06427 ; 06428 ;Logical disk D: (8" floppy)
090B B022 06423 DW Directory\$Buffer ;Shares same buffer as A: 090D 4409 06424 DW Floppy\$B\$Parameter\$Block 090F F023 06425 DW Dis\$\$C\$Workarea ;Private work area 0911 3C24 06426 DW Dis\$\$C\$Allocation\$Vector ;Private allocation vector 06427 ; 06428 ;Logical disk D: (8" floppy)
090D 4409 06424 DW Floppy\$8\$Parameter\$Block 090F F023 06425 DW Disk\$C\$Workarea Private work area 0911 3C24 06426 DW Disk\$C\$Allocation\$Vector Private allocation vector 06427 ; 06428 ;Logical disk D: (8" floppy)
090F F023 06425 DW Disk\$C\$Workarea ;Private work area 0911 3C24 06426 DW Disk\$C\$Allocation\$Vector ;Private allocation vector 06427 ; 06427 ; 06428 ;Logical disk D: (8" floppy)
0911 3C24 06426 DW Disk\$C\$Allocation\$Vector ;Private allocation vector 06427 ; 06428 ;Logical disk D: (8" floppy)
06427 ; 06428 ;Logical disk D: (8" floppy)
06428 ;Logical disk D: (8" floppy)
I AGIO AEAO - AKAOO - BU - ElonnyE56Skeutable - Shares same skeu tante as At
0913 AE09 06429 DW Floppy\$5\$Skewtable ;Shares same skew table as A: 0915 0000000006430 DW 0,0,0 ;Reserved for CP/M
091B B022 06431 DW Directory\$Buffer ;Shares same buffer as A:

Figure 8-10. (Continued)

4409	06432		DW	Floppys	8\$Parameter	\$Block	;Same DPB as C:
0024	06433		DW		Workarea	-DIOCK	;Private work area
5824	06434		DW		Allocation\$	Vector	Private allocation vector
	06435						,
	06436				;Logical d	isk M:	(memory disk)
	06437	M\$Disk\$I	DPH:		• •		
0000	06438		DW	0		;	No skew required
	06439		DW	0,0,0			Reserved for CP/M
B022	06440		DW		ry\$Buffer		
5409	06441		DW		Parameter \$ B	lock	
0000	06442		DW	0		1	Disk cannot be changed, therefore
	06443					;	no work area is required
7A24	06444		DW	M\$Disk\$4	Allocation\$	Vector	
	06445	;					
		;					
		;	Equates	for disk	k parameter	block	
		;					
		;	Disk Typ	es			
		<u>,</u>	_		_		
2							nini floppy
			3		2 ;8	" flopp	y (SS SD)
-				EQU	з ;M	emory c	11 SK
		;					
		-	Blocking	a/deplock	king indica	tor	
			lankir-	FOU	100000000	-	Contan alon > 100 butan
			10CK108	EWU	1000#0000B	,	Sector size > 128 bytes
			Diek		looks		
			DISK Par	aneter C	JIOCKS		
			5 1/4" -	nini flo-			
			5 1/4 1		5 P Y		
		•			. 5	vtra bu	te prefixed to indicate
							yre prefixed to indicate ype and blocking required
			DR	Floppy#S			
					: Ti	he para	meter block has been amended
					, .,	to ref	lect the new layout of one
					í	track	per diskette side, rather
					í	than v	iewing one track as both
					;	sides	on a given head position.
					• I ·		ilso been adjusted to reflect
	06615				;	one "n	ew" track more being used for
	06616				;		<pre>?/M image, with the resulting</pre>
	06617				;		in the number of allocation
	06618				;		and the number of reserved
	06619				;	tracks	· ·
		Floppy\$5					
							e sectors per track
				-			
							allocation block number
			2.11				of directory entries - 1
							for reserving 1 alloc. block
							ile directory
							anged work area size
0300			DW	ى	; N	umber (of tracks before directory
			Standar	4 9" 51	DD Y		
		,	scandard	1 9 F 10p		vtra bi	te prefixed to DPB for
							ersion of the BIOS
02			DB	Floppy			is disk type and the fact
~_							b deblocking is required
		Flopovs	***	ter\$Block			
1400		1 10000 000				ectors	per track
03	06640		DB	3		lock st	
07	06641		DB	7		lock ma	
00	06642		DB	ó		xtent a	
	06643		DW	242			allocation block number
F200							
F200 3F00	06644		DW		• N	umber r	of directory entries - 1
3F00	06644		DW DB	63 1100\$000			of directory entries - 1 for reserving 2 alloc. blocks
3F00 C0	06645		DW DB DB	1100\$000 0000\$000	00B ;B	it map	of directory entries - 1 for reserving 2 alloc. blocks ile directory
3F00			DB	1100\$000	00B ;B 00B ;	it map for fi	for reserving 2 alloc. blocks
	0000000000 B022 5409 0000 7A24 = = = = = = 31	06436 06437 0000 06438 8022 06440 5409 06441 0000 06442 06443 7A24 06444 06445 06447 06447 06447 06445 06445 06445 06455	04436 00000000000439 8022 06440 5409 06441 0000 06442 006443 7 724 06444 7 06446 7 06447 7 06448 7 06447 7 06448 7 06449 7 06450 7 06451 Floppy% 06451 Floppy% 06456 7 06456 7 06456 7 06456 7 06457 Need%Det 06457 8 06457 7 06456 7 06600 7 0600 7 0600 7 7 000 0600 7 7 000 0000 7 7 000 0000 7 7 000 0000 7 7 00000 7 7 00000000	06436 06437 M\$Disk\$DPH: 0000 06438 DW 0000000000439 DW 8022 06440 DW 5409 06441 DW 0000 06443 DW 06443 T 724 06444 DW 06445 ; 06446 ; 06447 ; Equates 06446 ; 06447 ; Equates 06448 ; 06445 ; 06450 ; 06450 ; 06451 Floppy\$5 = 06451 Floppy\$5 = 06457 Need\$Deblocking 06456 ; = 06457 Need\$Deblocking 06600 ;# 06600 ; 06600 DB 06600 DB 0600 DB 000 DB 000 DB 000 DB 000 DB 000 DB 000 DB 000 DB	06433 06437 0000 06438 DW 0,0,0 8022 06440 5409 06441 DW M*Disk* 0000 06442 7424 06444 7424 06444 000 06445 ; 06446 ; 06445 ; 06446 ; 06447 ; Equates for dis 06448 ; 06448 ; 06448 ; 06449 ; Disk Types 06450 ; 06451 Floppy* 06452 Floppy* 06455 ; Blocking/deblocking 06455 ; Blocking EQU 06455 ; 06455 ; 06455 ; 06600 ;* 06600 ;* 06610 ;* 06611 ;* 06612 ;* 06613 ;* 06613 ;* 06613 ;* 06614 ;* 06614 ;* 06615 ;* 06614 ;* 06615 ;* 06616 ;* 06616 ;* 06617 ;* 06618 ;* 06618 ;* 06618 ;* 06619 ;* 06619 ;* 06619 ;* 06619 ;* 06610 ;*	06435 yLogical d 0000 06437 M\$Disk\$DPH; 0000 06439 DW 0,0 0000 06437 DW 0,0,0 8022 06440 DW M\$Disk\$Parameter\$E 0000 06442 DW 0 06445 ; 0 0 7A24 06444 DW M\$Disk\$Parameter\$E 06446 ; 0 0 06446 ; 0 0 06445 ; 0 0 06445 ; 0 1 06455 Floppy\$5 EQU 1 06455 Plocking/deblocking indica 0 06456 ; 0 1000\$0000B 06457 Need\$Deblocking EQU 1000\$0000B 06458 ; 0 0 06600 ;# 0 0 06601 ; 1000\$0000B ; 06612 ; ; 1000\$0000B <td>06436 ;Logical disk M: 06437 M#Disk#DPH; 0 0000000006439 DW 0,0,0 5409 06441 DW MBDisk#Parameter#Block 06443 DW M*Disk#Parameter#Block 06443 DW M*Disk#Allocation#Vector 06444 DW M*Disk#Allocation#Vector 06445 ; 0 06446 ; 0 06447 Equates for disk parameter block 06448 ; 06447 Disk Types 06450 ; 06451 Floppy#5 06452 Floppy#5 06453 H#Disk 06455 Blocking/deblocking indicator 06455 Blocking EQU 100090000B 06600 ;# 06601 ; isk rank 06602 DB Floppy#5 + Need\$Delocking 06603 ; ; 06604 5 ; 06605 ; ; 0</td>	06436 ;Logical disk M: 06437 M#Disk#DPH; 0 0000000006439 DW 0,0,0 5409 06441 DW MBDisk#Parameter#Block 06443 DW M*Disk#Parameter#Block 06443 DW M*Disk#Allocation#Vector 06444 DW M*Disk#Allocation#Vector 06445 ; 0 06446 ; 0 06447 Equates for disk parameter block 06448 ; 06447 Disk Types 06450 ; 06451 Floppy#5 06452 Floppy#5 06453 H#Disk 06455 Blocking/deblocking indicator 06455 Blocking EQU 100090000B 06600 ;# 06601 ; isk rank 06602 DB Floppy#5 + Need\$Delocking 06603 ; ; 06604 5 ; 06605 ; ; 0

Figure 8-10. (Continued)

	06649	;		
	06650	; M\$Disk		
	06651	;		
	06652			;The M\$Disk presumes that 4 x 48K memory
	06653			; banks are available. The following
	06654			; table describes the disk as having
	06655			; 8 tracks: two tracks per memory bank
	06656			; with each track having 192 128-byte
	06657			sectors.
	06658			; The track number divided by 2 will be
	06659			; used to select the bank
0953 03	06660	DB	M\$Disk	; Type is M\$Disk, no deblocking
0/00 00	06661	M\$Disk\$Paramete		, type is hubisk, no debiocking
0954 C000	06662	nu nu	192	;Sectors per "track". Each track is
0/04 0000	06663	DW	172	; 24K of memory
0956 03	06664	DB	3	; 24K of memory ;Block shift (1024 byte allocation)
0957 07	06665	DB	7	Block mask
0958 00	06666	DB	0	
0959 C000	06667	DW	192	;Extent mask
0958 3F00		DW	63	Maximum allocation block number
	06668			;Number of directory entries -1
095D C0	06669	DB	1100\$0000B	Bit map for reserving 2 allocation blocks
095E 00	06670	DB	0000\$0000B	; for file directory
095F 0000	06671	DW	0	Disk cannot be changed, therefore no
	06672			; work area
0961 0000	06673	DW	0	;No reserved tracks
	06674	;		
0004 =	06675	Number\$of\$Logid	al\$Disks	EQU 4
	06676	;		
	06800	;#		
	06801	;		
	06802	SELDSK:	;Select disk i	
	06803		;C = 0	for drive A, 1 for B, etc.
	06804		;Retur	n the address of the appropriate
	06805			k parameter header in HL, or 0000H
	06806		; if	the selected disk does not exist.
	06807		;	
0963 210000	06808	LXI	н, о	;Assume an error
0966 79	06809	MOV	A,C	Check if requested disk valid
	06810		-	
0967 FEOC	06811	CPI	'M' - 'A'	;Check if memory disk
0969 CA9509	06812	JZ	SELDSK\$M\$Disk	;Yes
	06813			• • • • •
096C FE04	06814	CPI	Number\$of\$Logi	cal\$Disks
096E D0	06815	RNC		Return if > maximum number of disks
	06816	1		
096F 322D0A	06817	STA	Selected\$Disk	;Save selected disk number
	06818			;Set up to return DPH address
0972 6F	06819	MOV	L,A	Make disk into word value
0973 2600	06820	MVI	H.O	
	06821			;Compute offset down disk parameter
	06822			; header table by multiplying by
	06823			; parameter header length (16 bytes)
0975 29	06824	DAD	н	; Parameter neader length (16 bytes) ;*2
0976 29	06825	DAD	H	;*4
0977 29	06826	DAD	н	:*8
0978 29	06827	DAD	н	;*16
0979 11F309	06828	LXI	D,Disk\$Paramet	
0979 11E308 097C 19	06829	DAD	D,DISK#Faramet	;DE -> appropriate DPH
097D E5	06830	PUSH	H	Save DPH address
UVID EU			п	JOAVE DEN AUURESS
	06831 06832	;		Annae diek navanates blaat ta
				Access disk parameter block to
	06833			; extract special prefix byte that
	06834			; identifies disk type and whether
	06835 06836			; deblocking is required
097E 110A00	06836	LXI	D 10	7 .Cot DPD pointax officit in DOM
			D, 10	;Get DPB pointer offset in DPH ;DE -> DPB address in DPH
0981 19	06838	DAD	D	
0982 5E	06839	MOV	E,M	;Get DPB address in DE
0983 23	06840	INX	H	
0984 56	06841	MOV	D, M	
0985 EB	06842	XCHG		;DE -> DPB
	06843			
	06844	SELDSK\$Set\$Disk		
0986 2B	06845	DCX	н	;DE -> prefix byte
0987 7E	06846	MOV	A, M	;Get prefix byte
0988 E60F	06847	ANI	OFH	;Isolate disk type
0700 EOVF	06847	MN 1	VI-11	, I SUI ALE UI SK LYPE

098A	32360A	06848		STA	Selected\$Disk	\$Type	;Save f	or use in low level driver
0980		06849		MOV	A,M		other co	py of prefix byte
098E		06850		ANI	Need\$Deblocki	ng	;Isolat	e deblocking flag
0990	32350A	06851		STA	Selected\$Disk			or use in low level driver
0993		06852		POP	н	;Recove	r DPH po	inter
0994	C9	06853		RET				
		06854		Hens - I				
0005	212309	06855 06856	SELDSK\$	LXI	H.M\$Disk\$DPH			: selected 1 correct parameter header
	C38609	06856		JMP	SELDSK\$Set\$Di	sksTvpe		normal processing
0770	000007	06858	;	0.11	0220011001101	5	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	the man processing
		07000	;*					
		07001	3					
		07002	;	Set log	ical track for	next read	l or writ	e
		07003	;					
		07004	SETTRK:					
099B		07005		MOV	Н, В	;Select	ed track	in BC on entry
0990		07006		MOV	L,C			
	222E0A	07007		SHLD	Selected\$Trac	K ;Save f	OF IOW I	evel driver.
09A0	.,	07008 07009		RET				
		07100	; ;#					
		07100	;*					
		07102	÷	Set loc	ical sector fo	r next rea	d or wri	te
		07103	;					
		07104	;					
		07105	SETSEC:			;Logica	al sector	· in C on entry
09A1		07106		MOV	A,C	_	_	
	32300A	07107		STA	Selected\$Sect	or ;Save f	or low 1	level driver
09A5	C9	07108		RET				
		07109	;					
		07200	;#					
		07201	1	Cat di			ware for	payt read or unite
		07202 07203	;	Set 019	K DAM (INPUt/U		ness for	next read or write
0946	0000	07203	; DMA\$Add	ressi	DW O		;DMA ac	idress
V2H0	0000	07204	:				,	
		07206	SETDMA:			;Addres	s in BC	on entry
09A8	69	07207		MOV	L,C		o HL to	
09A9	60	07208		MOV	Н, В			
09AA	22A609	07209		SHLD	DMA\$Address	;Save f	or low 1	level driver
09AD	C9	07210		RET				
		07211	;					
		07300	;#					
		07301 07302	:	Tranel	ate logical sec	tor number	to shve	sical
		07302	;		iogical sec	tor number	to phys	
		07304	;	Sector	translation ta	bles		
		07305	;				the logi	ical sector number,
		07306	;					sector number.
		07307	;				•	
		07308	Floppy\$	5\$Skewt				contains four
		07309				byte secto		DI 1. 1. 540 1. 1
		07310		;	Physical 128t			Physical 512-byte
	00010203	07311		DB	00,01,02,03	;00,01,	02,03	0) 4)
09B2	10111213	07312		DB DB	16, 17, 18, 19	;04,05, ;08,09,		8)
	20212223 0C0D0E0F	07313 07314		DB	32,33,34,35 12,13,14,15	;12,13,		3)Head
	1C1D1E1F	07314		DB	28,29,30,31	;16,17,		7)0
	08090A0B	07315		DB	08,09,10,11	;20,21,		2)
	18191A1B	07317		DB	24,25,26,27	;24,25,		6)
	04050607	07318		DB	04,05,06,07	;28,29,		1)
	14151617	07319		DB	20,21,22,23	; 32, 33,		5)
		07320	,					
09D2	24252627	07321		DB	36,37,38,39	;36,37,	,38,39	0- 1
09D6	34353637	07322		DB	52,53,54,55	;40,41,		4]
09DA	44454647	07323		DB	68,69,70,71	;44,45	,46,47	8]
	30313233	07324		DB	48,49,50,51	;48,49		3] Head
	40414243	07325		DB	64,65,66,67	;52,53,		7] 1
	2C2D2E2F	07326		DB	44,45,46,47	;56,57		2]
	3C3D3E3F	07327		DB	60,61,62,63	;60,61		6]
	28292A2B	07328		DB	40,41,42,43	;64,65,		1] 5]
09F2	38393A3B	07329		DB	56,57,58,59	;68,69	,/0,/1	L C
		07330 07331	2					
		07331	Flopeve	8\$\$Skewt	ahle: star	ndard 8" Di	river	

Figure 8-10. (Continued)

		07333			01.02.0	3.04.05.	06,07,08	09.10	Logical sectors
09F6	01070D131			, DB			05,11,17		;Physical sectors
		07335		;		-,,,			,,
		07336		,	11, 12, 1	3, 14, 15,	16, 17, 18	, 19, 20	Logical sectors
00A0	090F15020	807337		DB			14,20,26		Physical sectors
		07338		;					
		07339		;	21,22,2	3,24,25,	26	Logica	l sectors
0A0A	1218040A1	007340		DB	18,24,0	4,10,16,	22	;Physica	al sectors
		07341							
		07400	;#						
		07401	;						
		07402	SECTRAN	:		;Transl	ate logi¢	cal secto	or into physical
		07403				;On ent	ry, BC =	logical	sector number
		07404				3	DE -1	> appropi	riate skew table
		07405				;			
		07406				;on exi			sector number
0A10		07407		XCHG				skew tabl	
0A11		07408		DAD	В				sector number
0A12		07409		MOV	L,M				ector number
	2600	07410		MVI	н, о		;Make in	nto a 16-	-bit value
0A15	C9	07411		RET					
		07412	;						
		07500	;#						
		07501	,						
		07502	; HOME:						
		07503	HUME:						cal disk to track O
		07504							neck must be made to see buffer has information in
		07505							
		07506							ten out. This is indicated by
		07507							uffer, that is set in the
		07508 07509				; debl	ocking co	pae.	
0414	3A2COA	07510		LDA		; ite\$Buff		• Check	if physical buffer must
0A19		07511		ORA	A	I (e + Buill	er		ritten to a disk
	C2200A	07512		JNZ	HOME\$No	turita		, De wi	Itten to a disk
	322B0A	07513		STA		\$Disk\$Bu	ffor	•No =0	indicate that buffer
VAID	OZZDUN	07514		314	Datavin	+0136+00			ow unoccupied
		07515	HOME\$No!	Write:				, 12 110	
0A20	0500	07516	1101124110	MVI	с,о			:Set to	track 0 (logically,
	CD9809	07517		CALL	SETTRK				ctual disk operation occurs)
0A25		07518		RET				,	
		07519							
		07520	;						
		07600	; #						
		07601	1	Data wr:	itten to	or read	from the	e mini-fl	loppy drive is transferred
		07602							ete track in length,
		07603	;	9 × 512	bytes.	It is de	clared at	t the end	d of the BIOS, and has
		07604	;	some sma	all amoui	nt of in	itializat	ion code	e "hidden" in it.
		07605	1						
		07606	1						o minimize the amount
		07607	;						and track
		07608	3				he physic		
		07609	7						CP/M "sector"
		07610	3						no disk access occurs
		07611	;	If a wri	ite requ	est occu	rs if and	the 128	-byte CP/M 'sector'
		07612	7	15 alrea	ady in th	ne physi	cal buffe	er, no di	isk access will occur,
		07613	7	UNLESS 1	the BDOS	indicat	es that i	ιτ 15 WP1	iting to the directory.
		07614	1	Directo	y write	s cause	an immedi	late writ	te to disk of the entire
		07615	1	track 1	n the ph	ysical b	utter.		
		07616 07617	7						
0800	_		7	ontPla-		EQU	2048		
0800		07618 07619		ion\$Bloc l\$Sec\$Per		EQU	2048		ed to reflect a "new"
0009	-	07620	FIJSICAL		₽11 CCR	240	,		to reflect a "new" the side of the
		07621						t disk	A TR OUT ONE SIDE OF CHE
0200	=	07622	Physical	\$Sector	Size	EQU	512		s the actual sector size
0200		07623	in, sica	. Poet tor	~~~~				the 5 1/4" mini-floppy diskettes
		07624						The B"	diskettes and memory disk
		07625							128-byte sectors
		07626							the physical disk buffer for th
		07627						5 1/4	4" diskettes
0004	=	07628	CPM\$Sect	Per\$Phys	sical	EQU	Physical		Size/128
0024		07629		Per\$Tra		EQU			sical*Physical\$Sec\$Per\$Track
1200		07630		er\$Track		EQU			r\$Track*Physical\$Sector\$Size
0003		07631	Sector#			EQU		Per\$Phy	
0002		07632		Bit\$Shif	t	EQU	2		<pre>>LOG2(CPM\$Sec\$Per\$Physical)</pre>
		···					-		· · · · · · · · · · · · · · · · · · ·

		07633 07634 07635 07636	\$			values handed over by the BDOS
		07635				
				1 ub		
		07434		, wit	en it cal	lls/ the write operation.
						i/unallocated indicates whether the
		07637				to write to an unallocated allocation
		07638				only indicates this for the first
		07639		, 12	8-byte se	ector write), or to an allocation block
		07640				lready been allocated to a file.
		07641		; The	BDOS also	o indicates if it wishes to write to
		07642		; th	e file di	irectory.
		07643		;	0	
0000 = 0001 =		07644 07645	Write\$Allocated Write\$Directory	EQU EQU	1	
		07645	Write\$Unallocated	EQU	2	<pre>s<== ignored for track buffering</pre>
0002 =		07647	Writesunallocated	EWO	2	TTTT IGNORED FOR TRACK Durrenting
0A26 00	~	07648	; Write\$Type:	DB	0	;Contains the type of write
UA20 U	v	07649	Writestype:	DB	Ū	; indicated by the BDOS
		07650	;			; Indicated by the bbos
		07651	7			
		07652	, In\$Buffer\$Dk\$Trk:		Varia	ables for physical sector currently
		07653	Inebuiter ebkett ki			Disk\$Buffer in memory
0A27 00	^	07654	In\$Buffer\$Disk:	DB	,	;) These are moved and compared
0A28 00		07655	In\$Buffer\$Track:	DW	ŏ	;) as a group, so do not alter
0420 00	000	07656	Int Dailter thi deki	2	•	; these lines
0A2A 00	0	07657	In\$Buffer\$Disk\$Typ	e: DB	0	Disk type for sector in buffer
	•	07658	1			
0A2B 00	0	07659	Data\$In\$Disk\$Buffe	r: DB	0	;When nonzero, the disk buffer has
	•	07660			-	; data from the disk in it
0A2C 00	0	07661	Must\$Write\$Buffer:	DB	0	Nonzero when data has been written
	-	07662				<pre>into Disk\$Buffer but not yet</pre>
		07663				<pre>written out to disk</pre>
		07664	;			
		07665	Selected\$Dk\$Trk:		;Varia	ables for selected disk, track and sector
		07666			; (Se)	lected by SELDSK, SETTRK and SETSEC)
0A2D 00		07667	Selected\$Disk:	DB	0	;) These are moved and compared
0A2E 00	000	07668	Selected\$Track:	DW	0	;) as a group so do not alter order
		07669				
0430 00	0	07670	Selected\$Sector:	DB	0	;Not part of group but needed here
		07671				
0A31 00	0	07672	Selected\$Physical\$	Sector: DB	0	Selected physical sector derived
		07673				; from selected (CP/M) sector by
		07674				<pre>shifting it right the number of</pre>
		07675				; bits specified by Sector\$Bit\$Shift
		07676 07677				
		07678	;			
0A32 00	0	07679	Disk\$Error\$Flag:	DB	0	Nonzero to indicate an error
UN32 U	~	07680	DISK#ENO # Tag.	00	v	; that could not be recovered
		07681				; by the disk drivers. The BDOS
		07682				; will output a "Bad Sector" message
0A33 00	0	07683	Disk\$Hung\$Flag:	DB	0	Nonzero if a watchdog timeout
	-	07684			-	i occurs
0258 =		07685	Disk\$Timer	EQU	600	Number of 16.66 ms clock ticks
		07686				; for a 10 second timeout
		07687	;			
		07688			;Flags	s used inside the deblocking code
		07689				
0A34 00	0	07690	Read\$Operation:	DB	0	Nonzero when a CP/M 128-byte
		07691				<pre>sector is to be read</pre>
0A35 00	0	07692	Selected\$Disk\$Deb]	lock: DB	0	Nonzero when the selected disk
		07693				; needs deblocking (set in SELDSK)
0A36 0	0	07694	Selected\$Disk\$Type	n DB	0	;Indicates 8" or 5 1/4" floppy or
		07695				##Disk selected. (set in SELDSK)
		07696	1			
		07800	; #			
		07801	, <u> </u>		00 (M	
		07802				tor specified by previous calls
		07803				Sector. The sector will be read
		07804	•	aaress spec	171e0 1h	the previous Set DMA Address call.
		07805	3 Té vendine			using sectors larger than 128 bytes,
		07806	If reading deblocking	y Trom a 019	he used	using sectors larger than 128 bytes, to "unpack" a 128-byte sector from
		07807 07808		al sector.	ne used i	to unpack a izo-byte sector from
		07808	; the physic READ:	ai sector.		
0A37 3	100500	07809		elected\$Disk	SDeblock	Check if deblocking needed
		07810	DRA A	.**************	-Depidek	; (flag was set in SELDSK call)
OA3A B		0/811				y crawy was set an oppoint cars?

Figure 8-10. (Continued)

OA3B	CA2FOB	07812		JZ	Read\$No\$Deblock	;No, use normal nondeblocked
		07813 07814				deblocking algorithm used is such
		07815				at a read operation can be viewed
		07816				til the actual data transfer as though
		07817				was the first write to an unallocated
		07818				location block
	3E01	07819		MVI	A, 1	;Indicate that a read actually
0A40	32340A	07820		STA	Read\$Operation	; is to be performed
		07821				
0443	3E00 32260A	07822		MVI	A,Write\$Allocated	Fake deblocking code into believing
0443	3220UA	07823 07824		STA	Write\$Type	<pre>; that this is a write to an ; allocated allocation block</pre>
0648	C35C0A	07825		JMP	Perform\$Read\$Write	; allocated allocation plock ;Use common code to execute read
		07826	;			yose common code to execute read
		07900	;#			
		07901	;			the current DMA address to
		07902	;	the pre	viously selected disk,	track and sector.
		07903	;	- ·		
		07904	;			l have set register C to indicate
		07905 07906	7			is to an already allocated allocation
		07906	;			of the sector may be needed), or
		07907	;		directory (in which cas mediately).	se the data will be written to the
		07908	,	UISK 1 0		
		07910	;	Only wr	ites to the directory i	take place immediately. In all other
		07911	;	cases.	the data will be moved	from the DMA address into the disk
		07912	;	buffer,	and only be written ou	ut when circumstances force the
		07913	;	transfe	r. The number of physic	cal disk operations can therefore
		07914	;		ced considerably.	
		07915	;			
		07916	WRITE:			
	3A350A	07917		LDA	Selected\$Disk\$Deblock	
OA4E		07918		ORA		; (flag set in SELDSK call)
UA4r	CA2A0B	07919 07920		JZ	Write\$No\$Deblock	
0A52	AF	07921		XRA	۵	;Indicate that a write operation
	32340A	07922		STA	Read\$Operation	; is required (i.e NOT a read)
0A56	79	07923		MOV	A,C	Save the BDOS write type
0A57	E601	07924		ANI	1	but only distinguish between
		07925				s write to allocated block or
0A59	32260A	07926		STA	Write\$Type	; directory write
		07927	;			
		07928	7			
		08000 08001	;*			
		08002	; Perform	\$Read\$Wr	ite: :Common code i	to execute both reads and
		08003				128-byte sectors.
0A5C	AF	08004		XRA	A JAssur	me that no disk errors will
0A5D	32320A	08005		STA	Disk\$Error\$Flag ; occ	cur
		08006				
	3A300A	08007		LDA		ert selected 128-byte sector
0A63		08008		RAR	; int	to physical sector by dividing by 4
0A64		08009		RAR	3FH :Remov	us any unumbed bits
0465	22310A	08010 08011		ANI STA		ve any unwanted bits
040/	32310A	08011		SIM	Selected\$Physical\$Sect	.ur
0444	212B0A	08012		LXI	H,Data\$In\$Disk\$Buffer	; Check if disk buffer already has
0A6D	7F	08014		MOV	A,M	; data in it
0A6E		08015		MVI	M, 1	; (Unconditionally indicate that
011012		08016				; the buffer now has data in it)
0A70	B7	08017		ORA	A	;Did it indeed have data in it?
	CA870A	08018		JZ	Read\$Track\$into\$Buffer	
		08019				track into the buffer
		08020			;	
		08021				buffer does have a physical track
					; in	it. Check if it is the right one
		08022			;	
		08022 08023				a Character and Assessed and the state of the state
	112704	08022 08023 08024		LXI	D, In\$Buffer\$Dk\$Trk	;Check if track in buffer is the
0A77	212D0A	08022 08023 08024 08025		LXI	H,Selected\$Dk\$Trk	; same as that selected earlier
0A77 0A7A	212DOA CDE10A	08022 08023 08024 08025 08025		LXI CALL	H,Selected\$Dk\$Trk Compare\$Dk\$Trk	; same as that selected earlier ;Compare ONLY disk and track
0A77 0A7A	212D0A	08022 08023 08024 08025 08025 08026 08027		LXI	H,Selected\$Dk\$Trk	; same as that selected earlier
0A77 0A7A	212DOA CDE10A	08022 08023 08024 08025 08025 08026 08027 08028		LXI CALL	H,Selected\$Dk\$Trk Compare\$Dk\$Trk	; same as that selected earlier ;Compare ONLY disk and track ;Yes, it is already in buffer
0A77 0A7A	212DOA CDE10A	08022 08023 08024 08025 08025 08026 08027		LXI CALL	H,Selected\$Dk\$Trk Compare\$Dk\$Trk	; same as that selected earlier ;Compare ONLY disk and track
0A77 0A7A 0A7D	212DOA CDE10A	08022 08023 08024 08025 08026 08027 08028 08029		LXI CALL	H,Selected\$Dk\$Trk Compare\$Dk\$Trk	; same as that selected earlier ;Compare ONLY disk and track ;Yes, it is already in buffer ;No, it will have to be read in

0A83 B7 08032 ORA Imust be written out first 0A84 C4850B 08033 CAL Write4Physical iver, write it out 0A87 CDCC0 08033 CAL SetSinSDufferSDtSTrk ise in bud dirver variables from 0A87 CDCC0 08033 CAL SetSinSDufferSDtSTrk ise in bud dirver variables from 0A86 CDEA00 08030 CAL ReadSTrackEintoBuffer ise in bud dirver variables from 0A86 CDEA00 08040 CAL ReadSTrackEintoBuffer ise in chard dirver variables from 0A86 CDEA00 08040 CAL ReadSTrackEintoBuffer ise in chard dirver variables from 0A86 CDEA00 08040 Ital NathiniteSDuffer ise in chard dirver variables from 0A87 3A300A 08044 ital ise in chard dirver variables from ise in chard dirver variables from 0A87 3A300A 08049 LDA SelectedStector iDet selected sector number iConvert the selected from outfirt 0A87 3A300A 08049 LDA iselectedStector iDet selected sector number iConvert the selected from outfirt 0A88 6F 08055 DAD H iselected sector number setor iConvert the selected sector number setor 0A98 6F 08055 DAD H iselectedStector number setor iCont						
0A88 C4ESOB 08033 CN2 Write%Physical ives, write%t out 0A87 CDCE0A 08035 Rad%Track%into%buffer: Set%in%buffer: Set%in%buffer: 0A87 CDCE0A 08030 CALL Set%in%buffer: Set%in%buffer: 0A86 CDEA0B 08040 CALL Read%Physical if Wesd the track is in the buffer now 0A86 CDEA0B 08040 CALL Read%Physical if Wesd the track is in the buffer 0A86 CDEA0B 08040 CALL Read%Physical if Wesd the track into the buffer 0A86 CDEA0B 08040 CALL Read%Physical if Wesd the track into the buffer 0A86 CDEA0A 08040 Track%In%Buffer: if Setected track and if Convert the setected CP/M (128-byte) 0A91 SA00A 08047 B0047 if Wo LA the buffer 0A92 20 08052 DAD H if 2 Set%ected Sector into a relative address down 0A94 25 08053 DAD H if 2 Set%ected Sector into a relative address down 0A94 25 08055 DAD H if 2 Set%ected Sector indik buffer 0A947 20 <	0483	B7	08032	ORA	Δ	, must be written out first
OBC34 08035 08035Pead&Track&into*Buffer1 CALLSet in buffer variables from i selectid disk, track i selectid disk, track i selectid disk, track i selectid disk, track i buffer now theich track is in i buffer now the track in the buffer i convert the selected track and i buffer disk is already in the buffer i buffer now a relative address don i buffer now their selectid track is in i buffer now track is in the buffer i buffer now their selectid track is in the buffer i buffer now a relative address don i buffer now their selectid track is in i buffer now track is in the buffer i buffer now their selectid track is in the buffer i buffer now their selectid track is in the buffer i buffer now their selectid track is in the buffer i buffer now their selectid track is in the buffer i buffer now their selectid track is in the buffer i buffer now their selectid track is in the buffer i buffer now their selectid track is in the buffer i buffer later on in disk buffer i buffer later in disk buffer i buffer later in disk buffer i buffer later in the buffer i buffer later in the buffer i buffer later now their data is to be moved i buffer later in the buffer i buffer later now their buffer i buffer later in the buffer i buffer later now in the buffe					Write\$Physical	
OB035 Read#Track%intoBuffer: Set in buffer variables from i selected disk. track i corflact which track is in 1 OAB CDEOD OB037 i corflact which track is in 1 OAB CDEOD OB040 CALL Read#Physical Treack into the buffer i contents OAB AF OB041 KA A PRead the track and i contents PRead the track and i contents OAB AF OB042 STA Must&Write&Buffer i setor into a relative address down i the buffer OB044 Track#In@Buffer: OB045 i Selected*Sector iDet selected OF/M (128-byte) i the buffer i the buffer OAS AF OB045 DAD HA i Hallipi by bits OAS AF OB052 DAD HA i Hallipi by bits OAS AF OB053 DAD H i # 2 OAS AF OB055 DAD H i # 2 OAS AF OB056 DAD H i # 2 OAS AF OB057 DAD H i # 2 OAS AF OB056 DAD H i # 2 OAS AF OB056 DAD	•	0.2000				y test write it dat
0A87 CDCE0A 08035 CALL SetsInsBufferSDksTrk Set in buffer variables from				7 DeadeTracksint	of Duffor.	
08037 00484 CDEADB 08039 08039 08039 08040 CALL CALL CALL CALL CALL CALL CALL CALL		CRCEAA				ATuk Cat in huffey youishing from
00038 i to reflect which track is in 1 0ABA CDEAOB 08040 CALL Read#Physical iRead the track into the buffer 0ABD AF 08041 STA A issue the flag to reflect buffer 0ABD AF 08043 STA HustWriteSDuffer issue the flag to reflect buffer 08045 irackEinsBuffer: issue sleeted track and i disk is already in the buffer 08044 trackEinsBuffer: issue sleeted Sector into a reliative address down 0A91 3A300A 08059 LDA Selected#Sector flog select allow address down 0A92 20 08051 HVI H.0 ieit 7 bits select allow address down 0A92 20 08053 DAD H # 2 select allow address down 0A95 20 08055 DAD H # 2 select allow address down select allow address down 0A95 29 08055 DAD H # 2 select allow address of disk buffer 0A95 29 08055 DAD H # 2 select allow address set in SETDM call <t< td=""><td>UH6/ (</td><td>CUCEUM</td><td></td><td>CALL</td><td>Setainaburrerab</td><td></td></t<>	UH6/ (CUCEUM		CALL	Setainaburrerab	
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$						
OAAB ACDEADB 08004 CALL ReadSPhysical iRead the track into the buffer OABB S22COA 08042 STA HustSWriteSBuffer i contents is alreading track and 08044 TrackSInSBuffer: iselected track and icontents is alreading track 08045 TrackSInSBuffer: iselected track and icontents is alreading track 08046 Geods it the buffer Convert the selected Sector number icontents 0A91 3A300A 08049 LDA Selected Sector number icontents 0A92 2400 08053 DAD H is alreading icontents 0A92 2400 08053 DAD H is alreading icontents icontents 0A92 290 08053 DAD H is alreading icontents						
OABD AF 08041 XRA A iRest the flag to reflect buffer 0ABE 322COA 08043 ; rack\$In\$Buffer: ; Selected track and ; onitents 08043 ; rack\$In\$Buffer: ; selected track and ; onitents 08044 ; rack\$In\$Buffer: ; selected track and ; onitents 08045 ; onitents ; selected track and ; onitents ; onitents 08046 ; onitents ; selected track and ; onitents ; onitents 08047 ; onitents ; old track and ; onitents ; onitents 08047 ; old track ; old track and ; onitents ; onitents 08047 ; old track ; old track and ; old track and ; onitents 04052 ; old track ; old track and ; onitents ; onitents 04052 ; old track ; onitents ; iest and ; onitents 04052 ; onitents ; iest and ; iest and ; iest and 04052 ; onitents ; iest and <						
OARE 322C0A 08042 08044 08044 STA Tack\$In\$Buffer: 08044 receive for the selected track and otsk is already in the buffer romvert the selected CP/H (128-byte) 08046 OAPI 3A300A 08049 LDA Selected\$Sector r0st selected member the buffer relative address down the buffer OAPI 3A300A 08049 LDA Selected\$Sector r0st selected member the buffer relative address down the buffer OAPI 3A300A 08049 LDA Selected\$Sector r0st selected member the buffer relative address down OAPI 2A20 08053 DAD H r* 2 OAPS 220 08053 DAD H r* 2 OAPS 220 08055 DAD H r* 2 OAPS 220 08055 DAD H r* 2 OAPS 220 08050 LXI D.Disk*Buffer relative member * 128 OAA2 2B3 08064 XCH0 relative address set in SISTMA call OAA2 2A600 08065 LHLD DMA&Address relative method used OAA2 2A600 08065 LHLD DMA&Address relative method used OAA2 2A600					Read\$Physical	
08043 08045 08046 08047 0805 0805	OABD (AF			Α	
08044 08045 Track#In#Buffer: 08046 Selected track and is is a lready in the buffer (Convert the selected CP/M (128-byte) is ector into a relative address down is the buffer 0A91 3M300A 08046 08047 is is is a lready in the buffer (Convert the selected CP/M (128-byte) is ector into a relative address down is the buffer 0A91 3M300A 08045 LDA Selected\$Sector 0A92 20 08051 MOV He 0A92 20 08053 DAD H 0A92 20 08053 DAD H 0A92 20 08053 DAD H 0A92 20 08055 DAD H 0A92 21 08055 DAD H 0A92 22 08055 DAD H is 32 0A92 21 08056 LXI D.Disk\$Buffer ieddress of disk buffer 0A92 21 08056 LXI D.Disk\$Buffer ieddress set in SEDMA call 0A42 21 <	OA8E :	322C0A		STA	Must\$Write\$Buff	er ; contents
08045 idsk is already in the buffer 08047 ; Convert the selected (Sector into a relative address down) 0A91 3A300A 08049 0A91 3A300A 08049 0A94 67 08050 0A92 67 08053 0A92 72 08055 0A92 72 08056 0A92 72 08057 0A93 72 08050 0A94 7 08050 0A92 7 08050 0A93 2A4069 08066 08067 DE				;		
08046 rConvert the selected CP/M (128-byte) 0A91 3A300A 08049 LDA Selected#Sector into a relative address down in the buffer 0A92 4F 08050 MOV L,A rMultiply bils Selected#Sector into a relative address down in the buffer 0A92 2200 08051 MU L,A rMultiply bils Selected#Sector into a relative address down in the intervalue 0A92 2200 08052 DAD H right right right 0A92 29 08055 DAD H right right right right 0A92 29 08055 DAD H right rig				Track\$In\$Buffe	r i	
08047 resctor into a relative address down the buffer 0A91 3A300A 08049 LDA Selected@Sector Get selected@Sector The selected@Sector <						; disk is already in the buffer
OAPI 3A300 OB049 LA Selected%Sector The buffer OAP3 4SC OB050 MOV L,A Hulliply by 128 by shifting 16-bit value OAP3 2600 OB051 MVI H,O Ileft 7 bits OAP3 220 OB052 DAD H rs 2 OAP3 220 OB053 DAD H rs 2 OAP3 220 OB053 DAD H rs 2 OAP3 220 OB055 DAD H rs 2 OAP3 220 OB055 DAD H rs 2 OAP3 220 OB055 DAD H rs 2 OAP3 220 OB056 DAD D DisseBuffer OAAP 20 OB064 XCHG DisseBuffer Ide in base address set in SETDM call OAAP 20 OB066 WUI C, 128/8						
OA91 SA300A OB005 LDA Selected\$Sector Tot is select a setor number OA92 S OB005 HVU H,0 IHITIPID by 128 by shifting 16-bit value OA92 S OB005 DAD H I#2 OA92 S OB005 DAD H I#3 OA92 OB005 LIN D.Disk\$Buffer OB01 OB1 OAA2 OB064 XCHO IDE -> Sector in disk buffer OAA2 OAA3 OB065 LHLD DMA\$Address IOE TOMA address et in SETDM call OAA6 EB OB066 XCHO IDE -> DMA address et in set out of the is buffer OAA7 OB070 HVI C.12						
$\begin{array}{cccccccccccccccccccccccccccccccccccc$						
0A95 2600 08051 HVI H,0 rift 7 bits 0A97 25 08052 DAD H r# 4 0A98 25 08055 DAD H r# 4 0A97 25 08055 DAD H r# 4 0A98 25 08055 DAD H r# 16 0A97 25 08055 DAD H r# 32 0A97 25 08057 DAD H r# 44 0A97 25 08057 DAD H r# 464 0A97 26 08059 r D.Disk#Buffer :Get base address of disk buffer 0A92 17 08051 DAD D r# 128 :H-5 20+ts sector number * 128 0A95 11A40F 08064 XCH0 DB-5 4ector in disk buffer :Get DMA address in the solf :Get DMA address set in SETDMA call 0AA2 EB 08064 XCH0 DMA#Address in the faster method used :Get DMA address in the source data in and out of the 0AA3 260 08067 :H-5 sector in disk buffer :Get fast on fask buffer :Get fast on fask buffer 0AA7 0E10 08067 :H-5 sector in disk buffer :Get fast on fask bu					Selected#Sector	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	0A94 (6F	08050	MOV		Multiply by 128 by shifting 16-bit value
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	0A95 2	2600	08051	MVI	н,о	şleft 7 bits
0A99 29 08055 DAD H ; # 8 0A9A 29 08055 DAD H ; # 32 0A9C 29 08055 DAD H ; # 32 0A9C 29 08056 DAD H ; # 32 0A9C 29 08057 JAD H ; # 128 08057 J 0A9E 11A40F 08060 LXI D.DiskBuffer iGet base address of disk buffer 08063 HL -> 128-byte sector number # 128 08063 HL -> 128-byte sector in disk buffer 08063 OF J 08063 OF J 08064 XCH0 JD DMA&Address of the start is address in disk buffer 08065 LHLD DMA&Address rot DMA address set in SETDA call 08066 VCH0 JD A D D DM&Address is disk buffer 08067 OF DMA address is disk buffer 08067 OF JD A DA D H L -> sector in disk buffer 08067 OF JD A DA D DMA&Address is in and out of the 08067 OF JD A DA D OF JD A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D J D A DA D J D J D A DA D J D J	0A97 3	29	08052	DAD	н	j* 2
0A99 29 08055 DAD H ; # 8 0A9A 29 08055 DAD H ; # 32 0A9C 29 08055 DAD H ; # 32 0A9C 29 08056 DAD H ; # 32 0A9C 29 08057 JAD H ; # 128 08057 J 0A9E 11A40F 08060 LXI D.DiskBuffer iGet base address of disk buffer 08063 HL -> 128-byte sector number # 128 08063 HL -> 128-byte sector in disk buffer 08063 OF J 08063 OF J 08064 XCH0 JD DMA&Address of the start is address in disk buffer 08065 LHLD DMA&Address rot DMA address set in SETDA call 08066 VCH0 JD A D D DM&Address is disk buffer 08067 OF DMA address is disk buffer 08067 OF JD A DA D H L -> sector in disk buffer 08067 OF JD A DA D DMA&Address is in and out of the 08067 OF JD A DA D OF JD A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D A DA D H L -> sector in disk buffer 08077 OF JD A DA D J D J D A DA D J D J D A DA D J D J			08053	DAD	н	ş¥ 4
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	0A99	29		DAD	н	
OA68 29 O8055 DAD H :# 32 OA9C 29 O8057 DAD H :# 128 OA9D 29 O8058 DAD H :# 128 OA9D 29 O8050 LXI D.Disk#Buffer :feet base address of disk buffer OAA1 19 O8061 DAD D :faddress in disk buffer OAA3 2AA609 O8064 XCHG :pE -> Sector in disk buffer OAA6 EB O8064 XCHG :pE -> DMA address is to in SETDMA call OAA7 0E10 O8069 MVI C,128/8 :pEcause of the faster method used 0AA7 0E10 O8069 MVI C,128/8 :pEcause of the faster method used 0AA7 0E10 O8067 :pE -> DMA address :pecause of the faster method used 0AA7 0E10 O8067 :pE -> DMA address :pecause of in disk buffer 0AA7 OE070 :pE -> DMA address :pecause of in disk buffer 0AA6 O8071 :pE -> DMA address :pecause 0AA0 CBO70 CA :poor count :pecause						
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	049B	29				
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	0A9C	29				
OASE1D. Disk®Buffer1Get base address of disk bufferOAA11908060LXID. Disk®Buffer1Get base address of disk bufferOAA11908062DADD1HL -> 128-byte sector number * 128OAA208063XCHG1DE -> 128-byte sector number startOAA32AA60908065LHLDDMA\$Address1DE -> sector in disk bufferOAA6EB08066XCHG1DE -> sector in disk bufferOAA70806811Assume a read operation, so08068080671DE -> sector in disk buffer0AA708069MVIC,128/8080701Because of the faster method used080711be 1/8 of normal080731be 1/8 of normal080741be 1/8 of normal080751C = loop count080761DA080771DE -> DMA address08078LDARead®Operation0AA208079ORA0AA308079DRA0AA41of the buffer0AA508083INR0AB5CDF80A0808308085XCHG10AB508086STA0AB5CDF80A080970AB5CDF80A080970AB508089INR0A10AB5080890AB5080970AB5080980AB5080990AB508099<	0A9D	29			н	
OAAE1140F08060LXID. Disk#BufferIGet base address of disk bufferOAAI1908062idd on sector number # 128OBAC208063i address in disk bufferOAA2 EB08064XCHGiDE -> sector in disk bufferOAA2 EB08065LLHLDMA&AddressiDE -> sector in disk bufferOAA6 EB08066XCHGiDE -> DMA address set in SETDMA callOAA7 0E1008069HVIC,128/8iBecaus of the faster method used08071it move data in and out of theit disk buffer08073it disk bufferit disk buffer0807408076it disk buffer08075it disk bufferit disk buffer0AA7 3A340A08076It address0AA7 3A340A08077it disk buffer0AA7 3A340A08078LDA0AA7 3A340A08079ORA0AAit disk bufferit disk buffer0AA8it disk buffer0AA9 3A340A08079ORA0AAit disk buffer0AA8080790AA8it disk buffer0AA9080800AB0 3C080800AB1 22COA080880AB2080850AB5080850AB5080850AB5080860AB5080860AB5080860AB5080860AB5080860AB5080860AB5080860AB5080860B08508085						
OAA1 19 OBO61 DAD D iAdd on sector number * 128 OAA2 EB 08063 iffer > 128-byte sector number start ; address in disk buffer OAA2 EB 08063 LHLD DMA\$Address iffer > 128-byte sector in disk buffer OAA6 EB 08065 LHLD DMA\$Address iffer > 08064 OAA6 EB 08066 iffer > 08067 iffer > 08067 iffer > 08067 OAA7 0E10 08069 MVI C,128/8 iffer > 180-buffer, (eight bytes moved per 08070 iffer > 128-buffer, (eight bytes moved per iffer < 08072	0A9E	11A40F			D.Disk\$Buffer	;Get base address of disk buffer
$\begin{array}{cccccccccccccccccccccccccccccccccccc$						
OAA2 EB OB064 XCH0 yDE → sector in disk buffer OAA5 2B OB064 XCH0 yDE → sector in disk buffer OAA6 EB OB066 yAE YDE → MA address OB067 DMA\$Address YDE → DMA address yEE OAA7 OE10 OB069 HVI C, 128/8 yBecause of the faster method used OAA7 OE10 OB071 JD (At this point, 00072 JD (At this point, 00073 JD (At this point, 00073 OB071 OB073 JD (At this point, 00073 JD (At this point, 00074 JD (At this point, 00074 OB074 JD (At this point, 00075 JD (At this point, 00074 JD (At this point, 00074 OAA7 OE10 OB075 JD (At this point, 00074 JD (At this point, 00074 OAA7 OE10 OB075 JD (At this point, 00074 JD (At this point, 00074 OAA7 OE10 OB075 JD (At this point, 00074 JD (At this point, 00074 OAA7 OE10 OB076 JD (At this point, 00074 JD (At this point, 00074 OAA7 OE10 OB076 JD (At this point, 00074 JD (At this point, 00074 OAA7 OE10 OB076 JD (At this point, 00076 JD (At this point, 00076	••••••	• ·			-	HI -> 128-byte sector number start
0AA2 EB 08064 XCH0 IDE -> sector in disk buffer 0AA3 2AA609 08065 LHL DMA&Address IDE -> DMA address IDE -> DMA address 0AA7 0E10 08068 YCH0 IDE -> DMA address IDE -> DMA address 0AA7 0E10 08069 MVI C,128/8 IBE-ause of the faster method used 08071 it to move data in and out of the it to move data in and out of the it to move data in and out of the 08073 it to move data in and out of the it to move data in and out of the it to move data in and out of the 08073 it to move data in and out of the it to move data in and out of the it to move data in and out of the 08073 it to move data in and out of the it to move data in and out of the it to move data in and out of the 08074 it to move data in and out of the inter ont in the it to move data in and out of the it to move data in and out of the 08074 it to move data in and out of the inter ont in the count need only it to the out for indisk buffer 08075 it DE -> DMA address it DE -> DMA address it to the out for 08076 ORA it DE -> Sector in disk buffer it the is to into the <tr< td=""><td></td><td></td><td></td><td></td><td></td><td></td></tr<>						
OAA3 2A609 OAA6 EBO8065 08064 08067 08068 08067 08068 08070 08071 08072 08073 08073 08074 08075 08075 08075 08076DMA\$Address resume a read operation, so r DE -> DMA address resume a read operation, so r DE -> DMA address resume a fread superation, so resume a fread superation, so resume a fread superation resume a fread superation	0000	C 0		YCHG		
0AA6 EB 08065 XCHG ; Assume a read operation, so ; DE -> DMA address ; HL -> sector in disk buffer 0AA7 0E10 08069 MVI C,128/8 ; Because of the faster method used ; to move data in and out of the 08071 0AA7 0E10 08069 MVI C,128/8 ; Because of the faster method used ; to move data in and out of the 08071 08071 ; disk buffer, (eight bytes moved)per 08073 ; loop iteration) the count need only ; be 1/8 of normal 08074 08076 ; DE -> DMA address 08077 ; DE -> DMA address 08077 0AA7 3A340A 08078 LDA 0AA7 2B50A 08069 JNZ 0AA6 9 ORA A ; out of the buffer (write) 04061 0AA8 08078 LDA 0AB0 3C 08063 INR 0AB1 322C0A 08063 STA 0AB5 CDF80A 08065 XCHG 0AB5 CDF80A 08091 ; to (DE) 08092 ; HL -> DMA address ; Make DE -> sector in disk buffer 0AB5 CDF80A 08065 XCHG 0AB5 ; HL -> DMA address ; Make DE -> sector in disk buffer 0AB5 08093 ; HL -> DMA ad					DMARAddrees	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					DIRPHOUTESS	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	UAAO I	ED		XCH0		PE -> DMA address
OAA7 0E1008069MVIC,128/8#Because of the faster method used i to move date in and out of the 08071 08072 08073 08073 08074 08075 08075 08075 08077Because of the faster method used i to move date in and out of the i disk buffer, (eight bytes moved per i loop iteration) the count need only i be 1/8 of normal per loop count i DE -> DMA address i DE -> DMA address i DE -> DMA address i DE -> DMA address i out of the buffer (read) or into the i buffer (write) i Bodoff in the buffer (read) or into the i buffer (write) i Bodoff i buffer i buffer (write)OAA0 C2B50A08080 08080 08081JNZ DE -> DMA address i out of the buffer (read) or into the i buffer (write) i Bodoff i buffer i buffer force a writeOAB0 3C08083 08084 08085 08087 08086 08087 08088 08088 08089JNR A Set flag to force a write i HL -> DMA address i HL -> DMA addressOAB5 CDF80A08090 08091 08092 08089 08089 08089 08089Move\$8 i Move\$8 i Move\$8 i to (DE)OAB5 SA260A08094 08094 08097 08088 08089 08089 08089 08089 08089 08089 08089 0AB5 SA260AMove\$8 i fwrite\$Type if write to directory, write out it write\$Directory is to (DE)OAB5 SA260A08094 08097 08097 08098 0AC2 C0DB089 08099 0RA 0AC2 C0JEDA 08099 0CAA 0AC2 C0OAB5 A260A08099 0CA 0AC2 C00CA 08099 0CADRA A is\$Error\$Flag if Get error flag in case delayed write or is\$Error\$Flag if delayed write or read i feturn if delayed write or read i is\$Extror\$Flag if Check if any disk errors h						
08070 ; to move data in and out of the 08071 08072 ; disk buffer, (eight bytes moved/per 08073 08073 ; be 1/8 of normal 08074 08074 ; be 1/8 of normal 08074 08075 ; DE -> DMA address 08077 ; DE -> DMA address 08078 LDA 08079 QRA 0AAD C2B50A 08060 08062 ; DI -> sector in disk buffer 0AB0 3C 08081 08081 ; sut of the disk buffer 0AB0 3C 08083 INR 0AB1 322C0A 08084 STA 0AB2 5 CDF80A 08090 CALL 0AB5 CDF80A 08090 CALL 08091 ; beffer\$Move: ; Set flag to force a write 0AB5 CDF80A 08090 CALL 08091 ; to (DE) 08082 ; 0AB8 3A260A 08094 0AB9 08095 0AB8 5A260A 08094 0AB9 CALL 0AB8 5A260A 08094 0AB9 CPI 0AB8 SA260A				MUT	6 100/0	
08071 ; disk buffer, (eight bytes moved per 08072 08073 ; bop iteration) the count need only 08074 ; At this point, 08075 ; C = loop count 08076 ; DE -> DMA address 08077 ; DE -> DMA address 08077 ; DH -> sector in disk buffer 0AA0 08078 LDA 0AA0 22B50A 08080 08081 ; Out of the buffer (read) or into the 0AB0 3C 08083 0AB0 3C 08084 0AB0 3C 08085 0AB0 3C 08086 0AB0 3C 08085 0AB0 3C 08085 0AB1 322C0A 08086 08087 ; 08088 iffer\$Move: 0AB2 08090 0AB3 08090 0AB5 08097 0AB8 3A260A 08094 LDA	UAA/ 0	OEIO		MV1	C,128/8	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$; to move data in and out of the
$\begin{array}{cccccccccccccccccccccccccccccccccccc$			08071			
08074 08075 08076 08077; C = loop count ; DE -> DMA address ; HL -> sector in disk buffer0AA9 3A340A 0A0078LDA 08077Read\$Operation ; DE -> DMA address ; HL -> sector in disk buffer0AA9 3A340A 0AA0 C2B50A08077 08079DRA A A A; DE -> DMA is buffer (read) or into the puffer (write)0AA0 C2B50A 0AA0 C2B50A08080 08081JNZ DUFF*Move puffer\$Move; out of the buffer (read) or into the puffer (write)0AB0 3C 0AB1 322C0A08084 08084STA STA Must\$Write\$Buffer; Set flag to force a write potential stater on. ; HL -> DMA address ; HL -> DMA address0AB1 322C0A 08086 08086 08087 08088 08088 08089Buffer\$Move: puffer\$Move: 08089; Set flag to force a write ; of the disk buffer later on. ; HL -> DMA address0AB5 CDF80A08090 08092 08093CALL Write\$Type 0ABB FE01 0A8095Move\$8 ; Move\$8 ; ff write to directory, write out 0ABB FE01 0A8097 0ABB SA320A 08097Write\$Type ; Jf write to directory, write out ; Return if delayed write or read ; Return if delayed write or read ; Return if delayed write or read ; Writes, abandon attempt to write to director ; Wres, abandon attempt to write to director ; Wres, abandon attempt to write to director ; Write to director			08072			
08075 ; C = loop count 08076 ; DE -> DMA address 08077 ; HL -> Sector in disk buffer 0AA2 3A340A 08078 LDA 0AA2 57 08079 DRA 0AA2 57 08079 DRA 0AAC 57 08060 JNZ 0AAD C2550A 08080 JNZ 08062 ;Writing into buffer (write) 0AB0 3C 08083 INR 0AB1 322C0A 08084 STA 0AB4 EB 08085 XCHG 08086 ; ;HL -> Sector in disk buffer later on. 0AB5 CDF80A 08099 CALL 0AB5 CDF80A 08094 LDA 0AB8 3A260A 08094 LDA 0AB8 3A260A 08094 LDA 0AB8 FE01 08095 CPI 0AB8 3A260A 08097 ;RNZ 0AB8 3A260A 08097 RNZ 0AB8 FE01 08095 CPI 0AB8 3A260A 08097 RNZ 0AB8 3A260A 08097 RNZ 0AC0 C0 08097 RNZ <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
08076; DE -> DMA address0AA9 3A340A08077LDARead\$Operation; DE -> DMA address0AAC B708079DRAA; out of the buffer (read) or into the0AAC B708060JNZBuffer\$Move;; out of the buffer (write)0AA0 C2B50A08068JNZBuffer\$Move;; buffer (write)0AB0 3C080683INRA; set flag to force a write0AB0 3C080683INRA; set flag to force a write0AB1 322C0A080684STAMust\$Write\$Buffer; of the disk buffer later on.0AB4 EB08065XCHG; HL -> DMA address08086;08068;08087;08068; HL -> DMA address0AB5 CDF80A08090CALLMove\$8; Moves 8 bytes * C times from (HL)08093; to (DE)080920AB3 320A008094LDAWrite\$Type; If write to directory, write out0AB8 FE0108095CPIWrite\$Directory; buffer immediately0AB8 FE0108095CPIWrite\$Directory; flet error flag in case delayed write or0AC0 C008097RNZ; Return if delayed write or read;0AC1 B708099ORAA; Check if any disk errors have occured0AC2 C008100RNZ;;08100;;;;08101;;;08101;;;08101;; <td></td> <td></td> <td></td> <td></td> <td></td> <td>At this point,</td>						At this point,
08077 0AA9 3A340A08078 08078LDA LDA Read\$DperationHL -> sector in disk buffer0AAC 57 0AAD C2B50A08079 08080DRA A 08081; out of the buffer (read) or into the puffer (write)0AB0 3C 0AB0 3C08083 08083INR 08084A state; Set flag to force a write ; A must be 0 get here)0AB0 3C 0AB4 EB 0808608083 08086INR stateA state; Set flag to force a write ; A must be 0 get here)0AB0 3C 0AB4 EB 08086080863 stateINR stateA state; Set flag to force a write ; A must \$Write\$Buffer ; of the disk buffer later on. ; HL -> DMA address0AB5 CDF80A 0AB5 CDF80A08099 08091 08092CALL Write\$Type stateMove\$8 ; Move\$8 bytes * C times from (HL) ; to (DE)0AB8 3A260A 0AB5 3A260A08094 08097LDA Write\$Type iff write to directory, write out if write to directory, write out if write to directory, write out 0AB5 3A260A0AB8 3A260A 0AC0 C008097 08097RNZ RNZ iff write to directory, write or read iff write to read iff write to read iff write to read iff write or read iff write to read iff write or read iff write to read iff write to write or read iff write to write or read iff write to write or read iff write to write to director iff write to write to director <br< td=""><td></td><td></td><td></td><td></td><td></td><td>; L = 100p count</td></br<>						; L = 100p count
OAAP 3A340AOBO78LDA OBC79Read\$Operation;Determine whether data is to be moved ; out of the buffer (read) or into the putfer (write)OAAD C2E50AOBO80JNZBuffer\$Move; buffer (write) ; buffer (write)OAB0 3COBO83INR OB084; Gamas be 0 get here)OAB0 3COB084STA OB085Must\$Write\$Buffer; of the disk buffer later on. ; Make DE -> Sector in disk buffer ; HL -> DMA addressOAB5 CDF80AOB089Buffer\$Move: OB086; Moves 8 bytes * C times from (HL) ; to (DE)OAB5 CDF80A08094LDA Write\$Type; ff write to directory, write out Write\$Directory ; buffer immediatelyOAB5 A220A08095CPI Write\$Directory ; Return if delayed write or read 08097OAB5 A220A08097RNZ (DAB5 3A220A; FNZ (DAB5 CDF80AOAB5 CDF80A08097CALL (DB091Mite\$Type ; If write to directory, write out ; to (DE)OAB5 A220A08095CPI (DAB5 3A220AWrite\$Type (RNZ (DAB5 3A220AOAC0 C008097RNZ (RNZ (DAB5 SA20A; OB099OAC1 B708099ORA (NZA ; Check if any disk errors have occured (Yes, abandon attempt to write to director) ; Yes, abandon attempt to write to director)						
OAAC B7OBO79ORA 08060A; out of the buffer (read) or into the buffer (write) ; buffer (write) ; Writing into bufferOAAD C2B50AOBO60JNZBuffer\$Move ; Buffer\$Move ; Writing into buffer ; Writing into buffer ; Writing into bufferOAB0 3COB083INR 08084; Set flag to force a write ; of the disk buffer later on. ; Hake DE -> sector in disk buffer ; HL -> DMA addressOAB3 CDF80AOB085XCHG 08086 08087 ; OB088 08088if write\$Buffer ; HL -> DMA addressOAB5 CDF80AOB090 08092 08093CALL 08092 08093Move\$8 ; Move\$8 ; Move\$8 bytes * C times from (HL) ; Jf write to directory, write out ; HI => DMA addressOAB5 SCDF80A08094 08092 08093LDA Urite\$Type ; Return if delayed write or read ; Yes, abandon attempt to write to director ; Write to director ; Wes, abandon attempt to write to director ; Yes, abandon attempt to write to director						
OAAD C2B50A 08061OB060 08061JNZ URBuffer\$Move; buffer (write) ;Writing into bufferOAB0 3C 0AB0 3C08083INR 08082A STA;Set flag to force a writeOAB1 322C0A 0AB4 EB 0AB508084STA 08085Must\$Write\$Buffer ; of the disk buffer later on. ;Make DE -> Sector in disk bufferOAB5 CDF80A08097 08091 08092CALL 08092Move\$8 ; to (DE)OAB5 CDF80A08094LDA UD 08092Write\$Type ; to (DE)OAB5 A260A08094LDA UDA UR UR UR OAB5 A220AWrite\$Type ; ff write to directory, write out ; buffer immediatelyOAB5 A260A08097 08097RNZ ; Return if delayed write or read ; Return if delayed write or read ; Write to directory ; Return if delayed write or read ; Write to directory ; Return if delayed write or read ; Write to directory ; RNZ ; Return if delayed write or read ; Write, abandon attempt to write to directory ; Write to directory ; RNZ ; RNZ ; RNZ ; RNZ ; Return if delayed write or read ; Yes, abandon attempt to write to directory ; Yes, abandon attempt to write to directory ; Stateston						
08081;Writing into buffer08082;(A must be 0 get here)0AB0 3C08083INR0AB1 322C0A08084STA0AB4 EB08085XCHG08086;08087;08087;08088;08089Buffer\$Move:080890809008089CALL08089guffer\$Move:080890809008091;08083;08083;08084;08095CALL08097;08093;08083;08083;08084DA08095;08096itie\$Type1write to directory, write out0AB3 3A200A080940AD0080970AC0 C0080970AC1 B7080990AC1 B7080990AC2 C00810008101;					••	; out of the buffer (read) or into the
08082 ;(A must be 0 get here) 0AB0 3C 08083 INR A 0AB1 322C0A 08084 STA Must\$Write\$Buffer ; of the disk buffer later on. 0AB4 EB 08085 XCHG ; Must\$Write\$Buffer ; of the disk buffer later on. 0AB4 EB 08085 XCHG ; Make DE -> sector in disk buffer 08087 ; 08086 ; 08088 ; 08089 sector in disk buffer 08087 ; 08087 ; 08088 ; 08089 sector in disk buffer 08089 Buffer\$Move:	OAAD (C2B50A		JNZ	Buffer\$Move	; buffer (write)
OABO 3COBOB3INRA; Set flag to force a writeOAB1 322C0AOBOB4STAMust\$Write\$Buffer; of the disk buffer later on.OAB4 EBOBOB5XCHG; make DE -> Sector in disk bufferOAB5 CDF80AOB087;OAB5 CDF80AOB094CALLOAB5 SA260AOB094LDAOAB5 SA260AOB095CPIOAB5 A200AOB095CPIOAB5 A200AOB095CPIOAB5 A200AOB097RNZOAB5 A200AOB097RNZOAB5 A200AOB097RNZOAC0 C0OB097RNZOAC1 B7OB099ORAOAC2 C0OB100RNZOAC1 B7OB099ORAOAC2 C0OB100RNZOAC1 B7OB099ORAOAC2 C0OB100RNZOAC1 B7OB099ORAOAC2 C0OB100RNZOAC2 C0OB100OAC2 C0OB100OAC2 C0OB100OAC2 C0OB100OAC3 C0OB100 <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>						
OAB1 322C0A08084STA 08085Must\$Write\$Buffer: of the disk buffer later on. ;Make DE \rightarrow Sector in disk buffer ;Make DE \rightarrow Sector in disk buffer ;HL \rightarrow DMA address 0808608086; 08087;HL \rightarrow DMA address08086; 08088;HL \rightarrow DMA address08086; 08089Buffer\$Move: 080890AB5 CDF80A08090CALL 08091Move\$80AB5 CDF80A08094LDA08087; to (DE)08093; 080930AB5 3A260A08094LDA Urite\$Type0AB8 5A260A08094LDA its\$Error\$Flag0AB8 5A260A08096LDA Urite\$Type0AB8 760108097RNZ ; Return if delayed write or read ; Return if delayed write or read ; Return if delayed write or read ; 0AC2 C00AC1 B708099ORA RNZ (0810108101; t						
OAB1 322C0A08084STA 08085Must\$Write\$Buffer: of the disk buffer later on. ;Make DE \rightarrow Sector in disk buffer ;Make DE \rightarrow Sector in disk buffer ;HL \rightarrow DMA address 0808608086; 08087;HL \rightarrow DMA address08086; 08088;HL \rightarrow DMA address08086; 08089Buffer\$Move: 080890AB5 CDF80A08090CALL 08091Move\$80AB5 CDF80A08094LDA08087; to (DE)08093; 080930AB5 3A260A08094LDA Urite\$Type0AB8 5A260A08094LDA its\$Error\$Flag0AB8 5A260A08096LDA Urite\$Type0AB8 760108097RNZ ; Return if delayed write or read ; Return if delayed write or read ; Return if delayed write or read ; 0AC2 C00AC1 B708099ORA RNZ (0810108101; t	OABO :	30	08083	INR	Α	;Set flag to force a write
OAB4 EB OB085 XCHG ; Make DE -> sector in disk buffer OB086 ; HL -> DMA address 08087 ; OB087 ; 08086 ; HL -> DMA address OB087 ; 08087 ; HL -> DMA address OB089 Buffer\$Move: 08099 ; the image of the image			08084	STA	Must\$Write\$Buff	er ; of the disk buffer later on.
08086 ; HL -> DMA address 08087 ; 08088 ; 08089 Buffer\$Move: 0AB5 CDF80A 08090 08091 ; to (DE) 08093 ; 08083 ; 08094 LDA Write\$Type ;If write to directory, write out 0AB5 A220A 08093 0AB5 A320A 08094 DAB jst\$Error\$Flag 0AB1 A3320A 08095 0AC0 C0 08097 0AC0 B7 ;RNZ if Return if delayed write or read 0AC1 B7 08099 0AC2 C0 08100 08101 ;						;Make DE -> sector in disk buffer
08087 ; 08088 ; 08089 Buffer\$Move: 0AB5 CDF80A 08090 0AB5 CDF80A 08091 08091 ; to (DE) 08092 ; 0AB5 CDF80A 08090 CAB5 CDF80A 08091 08091 ; to (DE) 08092 ; 0AB8 SA260A 08095 0AB8 FE01 08095 CPI Write\$Type ;If write to directory, write out 0ABB FE01 08095 CPI Write\$Type ;If write to directory, write out oase delayed write or 0AB0 A3200A 08095 CPI Write\$Type ;Return if delayed write or read ;Return if delayed write or read 0AC1 B7 08099 ORA ;Check if any disk errors have occured 0AC2 08100 ;Yes, abandon attempt to write to director get						; HL -> DMA address
08088 ; 08089 Buffer\$Move: 0AB5 CDF80A 08090 CALL Move\$8 ;Moves 8 bytes * C times from (HL) 08091 ; to (DE) 08092 ; 08093 ; to (DE) 0AB5 RE01 08094 LDA Write\$Type ;If write to directory, write out 0AB5 RE01 08095 CPI Write\$Directory ; buffer immediately 0AB5 A3200A 08094 LDA Disk\$Error\$Flag ;Get error flag in case delayed write or 08093 ; 0AB5 A320A 08096 LDA Disk\$Error\$Flag ;Get error flag in case delayed write or 08098 ; 0AC0 C0 08097 RNZ 08098 ; 0AC1 B7 08099 DRA A ;Check if any disk errors have occured 08100 ;Yes, abandon attempt to write to director				;		
08089 Buffer\$Move: 0AB5 CDF80A 08090 CALL Move\$8 ; Moves 8 bytes * C times from (HL) 08091 ; to (DE) 08092 ; (DE) 08093 ; ; to (DE) 0AB5 3A260A 08094 LDA 0AB5 FE01 08095 CPI Write\$Directory ; buffer immediately 0AB3 A320A 08096 LDA 0AC0 CO 08097 RNZ ; Return if delayed write or read 0AC1 B7 08099 0AC2 CO 08100 RNZ ;Yes, abandon attempt to write to director						
OAB5 CDF80A 08090 CALL Move\$8 ;Moves 8 bytes * C times from (HL) 08091 ; to (DE) 08092 ; (DE) 08093 ; (DE) 0AB5 A260A 08094 LDA Write\$Type ;If write to directory, write out 0AB5 A320A 08095 CPI Write\$Tirectory ; buffer immediately 0ABD 3A320A 08096 LDA Disk\$Error\$Flag ;Get error flag in case delayed write or read 0AC0 C0 08097 RNZ ;Return if delayed write or read ; 0AC1 B7 08099 DRA A ;Check if any disk errors have occured 0AC2 C0 08100 RNZ ;Yes, abandon attempt to write to director						
08091 ; to (DE) 08092 ; 08093 ; 0AB8 3A260A 08094 LDA Write\$Type ; if write to directory, write out 0AB8 5A260A 08094 LDA Write\$Type ; If write to directory, write out 0ABB 5A320A 08095 CO 08097 RNZ ; 0AC1 B7 08099 0AC2 C0 08100 08100 ;	OAR5	CDESOA			Move\$8	:Moves 8 bytes * C times from (HL)
08092 08093 08093 0ABS 3A260A 08094 0ABB FE01 08095 0ABD 3A320A 08096 0ABD 3A320A 08096 0ABD 3A320A 08096 0AC0 C0 08097 RNZ 0AC1 B7 08099 0AC1 B7 08099 0AC2 C0 08100 RNZ 08100 RNZ 08100 RNZ 08100 7 08101 7 08101 7 0000 1 1 1 1 1 1 1 1 1 1 1 1 1						
08093 ; 0ABS 3260A 08094 LDA Write\$Type ;If write to directory, write out 0ABB FE01 08095 CPI Write\$Tpirectory; buffer immediately 0ABD 3A320A 08096 LDA Disk\$Error\$Flag; ;Get error flag in case delayed write or read 0AC0 C0 08097 RNZ ;Return if delayed write or read 0AC1 B7 08099 DRA A 0AC2 C0 08100 RNZ ;Yes, abandon attempt to write to director						•
OABS 3A260A 08094 LDA Write\$Type ;If write to directory, write out OABB FE01 08095 CPI Write\$Directory; buffer immediately OABD A320A 08096 LDA Disk\$Error\$Flag; Bet error flag in case delayed write or OAC0 C0 08097 RNZ ;Return if delayed write or read 08098 ;Return if delayed write or read 0AC1 B7 08099 ORA ;Check if any disk errors have occured 0AC2 C0 08100 ;Yes, abandon attempt to write to director						•
0ABB FE01 08095 CPI Write\$Directory ; buffer immediately 0ABD 3A320A 08096 LDA Disk\$Error\$Flag ;Get error flag in case delayed write or 0AC0 C0 08097 RNZ ;Return if delayed write or read 0AC1 B7 08099 ORA A ;Check if any disk errors have occured 0AC2 C0 08100 RNZ ;Yes, abandon attempt to write to directo 08010 ; ;Return if delayed write or read	OAP9	342604		I DA '	Write\$Tvpe	
OABD 3A320A 08096 LDA Disk\$Error\$Flag ;Get error flag in case delayed write or present of the second o					WritesDirectory	: buffer immediately
OACO CO 08097 RNZ ;Return if delayed write or read 08098 ; 0AC1 B7 08099 0RA ;Check if any disk errors have occured 0AC2 C0 08100 RNZ ;Yes, abandon attempt to write to directo 08101 ;					Disk\$Error\$F1ag	:Get error flag in case delayed write or read
08098 ; OAC1 B7 08099 ORA A ;Check if any disk errors have occured OAC2 CO 08100 RNZ ;Yes, abandon attempt to write to directo 08101 ;					STREET OF FILES	
OAC1 B7 08099 ORA A ;Check if any disk errors have occured OAC2 CO 08100 RNZ ;Yes, abandon attempt to write to directo 08101 ;	UNCO I			TNIAL		
OAC2 CO 08100 RNZ ;Yes, abandon attempt to write to directo 08101 ;	0401	87		ORA	<u>م</u>	Check if any disk errors have occured
08101 7						Yes, abandon attempt to write to directory
	VAUZ			RINZ		
				VD.4	•	
OAC4 322COA 08103 STA Must&WriteBuffer ; written out					mustawriteaBuff	er ; written out
0AC7 CDE50B 08104 CALL Write\$Physical ;Write buffer out to physical track					Write\$Physical	Write butter out to physical track
OACA 3A320A 08105 LDA Disk\$Error\$Flag;Return error flag to caller					Disk\$Error\$Flag	;Keturn error flag to caller
0ACD C9 08106 RET	OACD	C9				
08107 ;			08107	;		

Figure 8-10. (Continued)

00000 f 1 0ACE 3A20A 00011 StA LPA SelectedBisk 0ACE 3A20A 00012 LDA SelectedBisk 1 now residing in buffer 0ACE 3A20A 00013 STA InBuffertFisk 0ACE 3A20A 00013 STA InBuffertFisk 0AD 2A2EA 00113 STA InBuffertFisk 0AD 2A2EA 00110 Sta Sta 0AD 32AAA 00119 STA InBuffertFisk 0AED 02 00110 Sta Sta 0AED 02 00110 Sta Formares just the disk and track r pointed to by DE and HL 0AEI 0E03 00127 CompareDBitTrki rCompares just the disk and track (2) 0AEI 0E03 00127 CompareDBitTrki rDompare Just the disk and track (2) 0AEI 0E03 00127 CompareDBitTrki rDomparedDitTrki 0AEI 0E03 00127 CompareDBitTrki rDomparedDitTrki 0AEI 0E03 00127 CompareDBitTrki rDomparedDitTrki 0AEI 0E03 0012 CompareDBitTrki rDomparedDitTrki 0AEI 0E03 0013 COMP rDomparedDitTrki 0AEI 0E03 0132 COMP rDomparedDitTrki 0AEI 0E03 0132 COMP rDomparedDitTrki 0					
OBEID OAEL 322704Setetine/BuilfertBukeTrkiIndicate selected juk, track i now residing in bufferOAEL 322704OBEI3LDA STAInBuffertBuikki now residing in bufferOAEL 322704OBEI3LLDA SHLDSelected%Track InBuffertFrackAlso reflect disk typeOAED 322804OBEI5SHLDInBuffertFrack InBuffertFrackOAED 4202064OBEI5SHLDInBuffertFrack InBuffertFrackOAED 600OBEI5InBuffertFrack InBuffertFrackOAED 600OBEI5InBuffertFrack InBuffertBuskTypeiAlso reflect disk typeOAED 600OBEI5CompareBukTrki InternationiCompareSukTrki InternationOAEI 0003OBEI5CompareBukTrki InternationiCompareSukTrki InternationOAEI 0003OBEI5CompareBukTrki InternationiCompareSukTrki InternationOAEI 0003OBEI5CompareBukTrki InternationiCompareSukTrki InternationOAEI 0003OBEI5CompareBukTrki InternationiCompareSukTrki InternationOAEI 0003OBEI5CompareBukTrki InternationiCompareSukTrki InternationOAEI 0003OBEI5IDAYIDAY InternationOAEI 0003OBEI5IDAYInternationOAEI 0003OBEI5IDAYIDAY InternationOAEI 0003OBEI5IDAYIDAY InternationOAEI 0003OBEI5IDAY InternationInternationOAEI 10003OBEI5IDAY InternationIDAY InternationOAEI			;		
OACE 3A2DOA OBILI imported by the source and source a					
OACE Saces Description Description OADI 32200A 08112 LHLD SelectedDisk Description OADI 3220A 08117 SHLD InsBufferSlisk OADI 3220A 08117 SHLD InsBufferSlisk OADI 3220A 08117 STA InsBufferSlisk OADI 3220A 08117 STA InsBufferSlisk OAED 08121 FET - - 08122 : : CompareSlistFirkLoop: - 0AED 08127 CompareSlistFirkLoop: . Description 0AED 08137 CompareSlistFirkLoop: . Description 0AED 08130 DRVZ . . Theadon comparison if inequality found 0AES 10 08137 0AES 08137 0AES 08137 			Set\$In\$Buffer\$	Dk\$Trk:	;Indicate selected disk, track
OAD1 32270A 00113 STA InBufferSDisk OAD4 2A2ECA 00115 LHLD SelectedBTrack OAD7 22280A 00115 LHLD SelectedBDiskTrype (Also reflect disk type OADA 3A360A 00117 ET (CompareSDistTrype (Also reflect disk and track OAE0 CP 00123 ; (CompareSDistTrk: ; pointed to by DE and H. OAE1 0E03 (CompareSDistTrk: ; pointed to by DE and H. (CompareSDistTrk: OAE1 0E03 (CompareSDistTrk: ; pointed to by DE and H. (CompareSDistTrk: OAE1 0E03 (CompareSDistTrk: ; pointed to by DE and H. (CompareSDistTrk: OAE2 0E13 UDX D ; flet comparitor OAE3 0E13 UDX D ; flet comparitor OAE5 08133 UDX D ; flet comparitor OAE5 08133 UDX TREUTUR ; isota and pointer OAE5 08133 UDX C ; flet source byte OAE5 08134 NVX N ; these pointed at by DE OAE5<					; now residing in buffer
0AD4 2A2CO, 0AD7 222804 08114 08116 LHLD SelectedStrack 0AD4 2A2CO, 0AD7 222804 08116 08116 SHLD InSBUfferSTrack 0AD4 3A3CO, 0AD0 322A0A 08119 08123 STA InSBUfferSDiskType ;Also reflect disk type 0AD4 3A3CO, 0AE0 C9 08121 08123 RET : compares just the disk and track 0AE1 0E03 08126 HUT CompareSDisTrikLC9: : compares inth comparing 0AE1 0E03 08126 HVT r.dompares inth comparing : compares inth comparing 0AE2 1A 08126 LDAY r.dompares inth comparing : compares inth comparing 0AE2 0A 08137 LDAY r.dompares inth comparing : compares inth comparing 0AE5 0A 08131 LNX D : LPASt comparing : comparing 0AE5 0A 08133 LNX D : LPASt comparing : comparing 0AE5 0A 08133 LNY D : LPASt comparing : comparing 0AE5 0A 08134 NZ : rotace comparing : comparing 0AE5 0A 08134					
OADA 222EOA 08115 LHLD SelectedBTrack OADA 223EOA 08117 LDA SelectedBD1skType rAlso reflect disk type OADD 33350A 08117 LDA SelectedBD1skType rAlso reflect disk type OAED 08120 RET	0AD1 32270A		STA	In\$Buffer\$Disk	
OAD2 22280AOB116SHLDInsbuffersTrackOADA 3A360AOB117LDASelected\$Disk\$Type:Also reflect disk typeOADD 3220AOB120STAInsbuffersTisk\$Type:Also reflect disk typeOAED CPOB121RET:Compare\$Disk\$Tyk::Compare\$List the disk and trackOAED CPOB122::Compare\$DiskTyk::Fonited to by DE and HLOAE1 0E03OB126HVI C.3:Disk (1), track (2)OAE2 IAOB127LDAX D:Get comparitorOAE3 IAOB128LDAX D:Get comparitor paintOAE3 COOB130RT M:Get comparitor pointerOAE5 COOB131INX D:Update comparitor pointerOAE5 COOB133DCR C:Count down on loop countOAE5 COOB136:UPF Compare\$DistTrkkLoop:OAE5 COOB137:UPF Compare\$DistTrkkLoop:OAE5 COOB138:UPF Compare\$DistTrkkLoop:OAE5 COOB136:UPF Compare\$DistTrkkLoop:OAE5 COOB136:UPF Compare\$DistTrkkLoop:OAE5 COOB137:UPF Compare\$DistTrkkLoop:OAE5 COOB141 <td></td> <td>08114</td> <td></td> <td></td> <td></td>		08114			
OADA 33300 08117 LDA Selected\$DiskType ;Also reflect disk type OADD 322A0A 08119 STA InsBuffer\$DiskType ;Also reflect disk type OAED CP 08122 ; ;Compares Just the disk and track 08124 CompareBD(%Trk: ;Compares Just the disk and track 08125 OB127 ;Dointed to by DE and HL 08126 CompareBD(%Trk:Cop: ;Disk (1), track (2) 0AEI 0000 08127 CompareBD(%Trk:Cop: 0AEI 0000 08127 CompareBD(%Trk:Cop: 0AEI 0000 08137 RTX photometand 0AES 1A 08127 CompareBD(%Trk:Cop: ;Return (uith zero flag set) 0AES 02 08133 DCR ;Return (uith zero flag set) 0AES 02 08134 RZ ;Plisk (1), Track (2) 0AEI 0003 08144 Move\$Dk=Trk: ;Plisk (1), Track (2) 0AEI 0003 08144 Move\$Dk=Trk: ;Plisk (1), Track (2) 0AEI 0103 08144 Move\$Dk=Trk: ;Plisk (1), Track (2) 0AEI 02 08144 Move\$Dk=Trk: ;Plisk (1), Track (2) 0AEI 02 08145 Move			LHLD	Selected\$Track	
OADA 3A330AOBITSLDASelectedSDiskTypejAlso reflect disk typeOAED C208120TTOAED C208121TT0812308123TT08124CompareSDIStTrk:Tpointed to by DE and HL0AEI 0E0308125TT0AEI 108126CompareSDIStTrk:T0AEI 108126CompareSDIStTrk:T0AEI 108126CompareSDIStTrk:T0AEI 108127TTAbandon comparison if neguality found0AES 108131INX DT0AES 108132INX HT0AES 108133INX HT0AEA C2830A08135JMPFC0AEA C2830A08135JMPFCompareSDISTrkSLoop:0AEA C2830A08135JMPFCompareSDISTrkSLoop:0AEA C2830A08135JMPFCompareSDISTrkSLoop:0AEA C2830A08141MVI C.3T Disk (1), Track (2)0AEA C2830A08142MOVEDUESTrkSLoop:Those pointed at by DE0AEA C2830A08143MOV A.HT Get source byte0AEA C2830A08144STAX DT Store in destination0AEA C2850A08144STAX DT Store in destination0AEA C2850A08147DCR CJCOunt down on byte count0AF3 0D08147DCR CJCOunt down on byte count0AF3 0D08147DCR CJCOunt down on byte count0AF3 0D08147DCR CJCOunt d	0AD7 22280A		SHLD	In\$Buffer\$Trac	k
OADD 322A0AOB110STAInsBuffer%Disk*TypeOAE0CP08121RET08124CompareSDk%Trk::CompareSlust the disk and track08125HVIC.3tDisk (1), track (2)OAE108126HVIC.3tDisk (1), track (2)OAE308127CompareSDk%TrkLoop::Detained to by DE and HLOAE308127CompareSDk%TrkLoop::Detained to by DE and HLOAE308126HVIC.3:Disk (1), track (2)OAE308137CompareSDk%TrkLoop::Detained to by DE and HLOAE508138RZ:Detained to by DE and HLOAE508133INXD:Detained to by DE and HLOAE508133INXD:Detained to by DE and HLOAE508133RZ:Detained to by DE and HLOAE508133RZ:Detained to by DE and HLOAE608133RZ:Detained to by DE and HLOAE708137:Detained to by DE and PL:Detained to by DE and HLOAE708139:Detained to by DE and PL:Detained to by DE and PLOAE708139:Detained to by DE and PL:Detained to by DE and PLOAE708139:Detained to by DE and PL:Detained to by DE and PLOAE708139:Detained to by DE and PL:Detained to by DE and PLOAE708139:Detained to by DE and PL:Detained to by DE and PLOAE708143:Detained to the pointers:Detained to by DE and PLOAE708					
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					
OAED CP 08121 08122 (08122 08122 08123 08127 RET OAEL 0E03 08127 08127 : CompareSLikeTrk: 08126 : CompareSLikeTrk: 10 pointed to by DE and HL OAEL 0E03 08127 08127 HVI C.3 DISK (1), track (2) OAES BE 0454 BE 0455 CB 0455	OADD 322AOA		STA	In\$Buffer\$Disk	\$Type
08122 08124 08125 Compare\$Dk\$Trk: 08126 ; Compares just the disk and track pointed to by DE and HL 10 kt Disk (1), track (2) 0AE1 0003 08127 0453 IA Compare\$Dk\$Trk1cop: 08127 rDisk (1), track (2) 0AE4 BE 08127 0453 IA Compare\$Dk\$Trk1cop: 0455 C0 is comparion 0AE5 1A 08127 0455 C0 CMP M ; Compare with comparing pointer 0AE5 00 08130 RNZ ; Abandon comparing pointer 0AE5 01 08131 INX D ; Ubdate comparing pointer 0AE5 02 08133 DCR ; Return (with zero flag set) 0AE5 02 08134 RZ ; Noves the disk, track is variables pointed at by DE 0AE5 02000 08141 MVI C.3 ; Disk (1), Track (2) 0AE5 02000 08141 MVI C.3 ; Disk (1), Track (2) 0AE1 02003 08141 MVV A.M ; Get source byte 0AE1 02003 08142 INX I ; Count down on byte count 0AE1 02003 08144 TAX I ; Count down on byte count 0AE1 02014 08149					
08123 08124; Compares Just the disk and track ; pointed to by DE and HL 081250AE1 0E0308125 08125MIIC.30AE3 1A08126Compare\$D\$\$Tr\$Lop; 100115Tisk (1), track (2)0AE4 BE08127 08128INXFde tomparitor intequality found 0AE5 130AE5 1A08128INXTybeate comparitor intequality found 0AE5 000AE5 0008130RNZ 100412intequality found intequality found intequality found intequality found 0AE5 020AE5 0108135INXH:Update comparitor intequality found intequality found intequality found intequality found intequality found integravity found integravi	OAEO C9		RET		
08124 0AE1 0E03 Compare\$Dk\$Trk: 08125 ; Compare\$Just the disk and track i pointed to by DE and HL 015 (1), track (2) 0AE5 1A 0AE5			;		
ORI 00:00 Original Status Pointed to by DE and HL ORIZO ORIZO CompareBDK#TrkLoop: CompareBDK#TrkLoop: ORIZO Get comparitor ADAM ORIZO CompareBDK#TrkLoop: Comparison Fight comparitor ADAM ORIZO ORIZO Fight comparitor ADAM Investigation ORIZO ORIZO Fight comparison Investigation ORIZO ORIZO ORIZO Fight comparison ORIZO ORIZO ORIZO Fight comparitor ORIZO ORIZO ORIZO Fight comparitor ORIZO ORIZO ORIZO Fight comparitor ORIZO ORIZO Fight comparitor Fight comparitor ORIZO ORIZO Fight comparitor Fight comparitor ORIZO ORIZO Fight comparison Fight c			;		
OAEI 0E120 0E126 MVI C.3 fDisk (1), track (2) OAE3 1A 08127 COmpare Wirk Comparand The comparison OAE5 08130 NNX Phandon comparison if inequality found OAE5 08130 NNX Phandon comparison if inequality found OAE5 08133 NNX Phandon comparison if inequality found OAE5 08133 NNX Phandon comparison if inequality found OAE5 08133 DCR C ;Count down on loop count OAE5 08133 JMP Compare WirkstriksLoop Worksthe disk, track OAE1 08137 ; "Movesthe pointed at by HL to ; tose pointed at by HL to OAE1 08140 MVI C.3 ; JDisk (1), track (2) OAE1 08142 MOVEDINESTrikELoop: ; Movesthe disk, track OAE1 08144 STAX D ; Store in destination OAE1 08145 INX D ; Update pointers OAE1 08145 INX D			Compare\$Dk\$Trk	:	
ORE3 /r OAE3 IA Compares bitsTrksLoop; OAE4 BE ORE of the comparison of inequality found OAE5 C0 08130 RNZ ; Get comparison if inequality found OAE5 C0 08131 INX D ; Update comparison if inequality found OAE5 C0 08133 ICR C ; Update comparison if inequality found OAE5 C0 08134 ICR C ; Update comparison if inequality found OAE5 C0 08134 ICR ; Update comparison if inequality found OAE5 C3E30A 08135 UMP Compare #Dit#TrksLoop 0AE5 C0 08134 RZ ; Variables pointed at by IL 0AE1 C2E3E30A 08135 UMP Compare #Dit#TrksLoop 0AE5 TE 08137 ivariables pointed at by IL ; variables pointed at by IL 0AE1 OE03 08140 HVI C.3 ; Disk (1), Track (2) 0AE5 INX D ; Disk (1), Track (2) 0AF2 23 08145 INX D ; Undate pointers 0AF2 23 08147 DCR ; Count down on byte count <t< td=""><td></td><td></td><td></td><td></td><td>; pointed to by DE and HL</td></t<>					; pointed to by DE and HL
OAE3 IA08128LDAXD;Get comparitorOAE4 BE08129CMPH;Compare with comparindOAE5 C008130RNZ;Abandon comparind pointerOAE5 C108132INXH;Update comparind pointerOAE5 C208132INXH;Update comparind pointerOAE5 C308132INXH;Update comparind pointerOAE5 C800135UMPCompare#Dk%Trk%Loop0AE5 C808137;;0AE1 DE0308141NVIC.30AE1 DE0308141NVIC.30AE5 C308141NVIC.30AE5 C308141NVIC.30AE5 C308143MOV A.M;fet source byte0AE5 C308144INXD;flow on byte count0AF5 C308144INXD;flow on byte count0AF5 C308145INXH;Update pointers0AF5 C3EFOA08146INXH;Update pointers0AF3 C3EFOA08147IDCRc found down on byte count0AF3 C3EFOA08148RZ;Return if all bytes moved0AF3 C3EFOA08146;JPP anwetskitrk&Loop0AF3 C3EFOA08149JMPMove\$Dk%Trk&Loop0AF3 C3EFOA08149JPPjPP0AF3 C3EFOA08149JPP0AF3 C3EFOA08149JPP0AF3 C3EFOA08149JPP0AF3 C3EFOA08149JPP0AF3 C3EFOA08	OAE1 OEO3				;Disk (1), track (2)
OAE4 BE08129CMPH;Compare with comparison if inequality foundOAE5 CO08130RNZ;JUpdate comparison if inequality foundOAE5 CI08131INXD;JUpdate comparison if inequality foundOAE5 CI08133DCRC;Count down on loop countOAE5 CI08133DCRC;Count down on loop countOAE5 CI08134RZ;Return (with zero flag set)OAEA CISEOA08135UMPCompare BDk\$Trk\$Loop0814008141Frktcopiivariables pointed at by HL to08140NVIC.3;Disk (1), Track (2)0AEF 7E08144STAXD0AF5 1108145INXD0AF5 1208144STAXD0AF5 1308145INXH0AF5 1208146INXH0AF5 1308146INXH0AF5 1408147CCR;Count down on byte count0AF5 0208148RZ;Return if all bytes moved0AF5 0208148RZ;Return if all bytes in a block, C times, from0AF5 028F0A08130iC= number of 8-byte blocks to move08100;UP secution.E081010AF5 7E08315MOV A,M;Get byte from source0AF5 7E08316NXH0AF5 7E08315MOV A,M;Get byte from source0AF5 7E08316NXH0AF5 7E08315MOV A,M;Get byte from					
OAES 10 OBI30 RNZ ;Abandon comparison if inequality found OAES 13 OBI31 INX H ;Update comparison if inequality found OAES 010 OBI32 INX H ;Update comparison if inequality found OAES 010 OBI32 INX H ;Update comparison pointer OAES 02 OBI34 RZ ;Count down on loop count OAES 03 OBI34 RZ ;Return (with zero flag set) OAES 040 OBI35 ; ;those pointed at by HL OBI37 ; those pointed at by HL to ; OAED 0E03 OBI41 MVI C.3 ;Disk (1), Track (2) OAET 0E03 OBI45 INX D ;Update pointers OAET 0E03 OBI45 INX D ;Update pointers OAET 0E04 OBI45 INX D ;Update pointers OAET 0E0 OBI45 INX D ;Update pointers OAET 0E0 OBI46 IX ;Update pointers OAET 0E0 OBI46 IX ;Update pointers OAET 0E0 OBI46 IX					
OAE5 13 OB131 INX D jUpdate comparind pointer OAE7 23 OB132 INX H jUpdate comparind pointer OAE8 08 OB133 DCR C ;Count down on loop count OAE8 08 OB133 DCR C ;Return (with zero flag set) OAEA C3E30A OB135 JMP Compare\$Dk\$Trk\$Loop ;Return (with zero flag set) OAE1 02 OB141 MVI C.3 ;Disk (1), Track (2) OAED 0E03 OB142 Move\$Dk\$Trk\$Loop: ;Moves the disk, track OAED 0E03 OB141 MVI C.3 ;Disk (1), Track (2) OAED 0E03 OB144 MVI C.3 ;Disk (1), Track (2) OAE1 12 OB144 STAX D ;Everce byte OAF1 13 OB144 INX H ;Everce byte OAF3 0D OB147 DCR ;Count down on byte count OAF3 02 OB148 R2 ;Return if all bytes moved OAF3 03 OB149 JMP Move8bkTrk\$Loop <t< td=""><td></td><td></td><td></td><td>м</td><td></td></t<>				м	
OAE72308132INXH:Update comparand pointerOAE8008808133DCRC:Count down on loop countOAE908136RZ:Return (with zero flag set)OAE1008136::Return (with zero flag set)OAE1008137::Return (with zero flag set)08137::Return (with zero flag set)08138MoveBD(%Trk::MoveSthestrk:08139::Variables pointed at by HL to08140:Variables pointed at by DE0AE7:08143MoveBD(%Trk!Loop:0AE7:08143MoveBD(%Trk!Loop:0AE7:08144INX0AF7:08145:NX0AF7:08144INX0AF7:08145:NX <td< td=""><td></td><td>08130</td><td>RNZ</td><td></td><td>;Abandon comparison if inequality found</td></td<>		08130	RNZ		;Abandon comparison if inequality found
OAEB 0B008133DCRC; Count down on loop count (with zero flag set)OAEA C3E30A08135JMPCompare\$Dk\$Trk\$Loop0AEA C3E30A08135JMPCompare\$Dk\$Trk\$Loop08137;;'variables pointed at by HL to i those pointed at by DE0AED 0E0308141MVIC,30AED 0E0308141MVIC,30AED 0E0308141MVIC,30AED 0E0308142MOVE*Dk\$Trk\$Loop:0AEF 7E08142MOVE*Dk\$Trk\$Loop:0AEF 7E08143MOV0AF6 1208144STAX0AF7 1208144STAX0AF7 1208144STAX0AF7 1208145INK0AF7 1208146INK0AF7 1208146INK0AF7 1208146INK0AF7 1208146INK0AF7 1306145INK0AF7 14WOV0AF8 020if0AF8 020if0AF8 020if0AF9 12081450AF9 120830513Image 140AF9 120830514Move\$Uthe moves eight bytes in a block, C times, from08305 i(HL) to (DE). It uses "drop through" coding to speed08306 iEntry Parameters08307 iC = number of 8-byte blocks to move08308 iEntry Parameters08309 iImage 140AF9 12083150AF9 12083160AF9 1208316	0AE6 13		INX		
OAES C8OB134RZiPeturn (with zero flag set)OAEA C3E30A08135JHPCompare\$Dk\$Trk\$Loop08136::08137::08138Move\$Dk\$Trk\$Loop:08139::04140::0AED 0E0308141MVI C.30AEF 7E08142Move\$Dk\$Trk\$Loop:0AEF 7E08143MUV0AF2 3208144STAX D0AF2 3108146INX H0AF3 0D08147DCR C0AF5 C3EF0A081480AF3 0D081470AF5 C3EF0A081490AF5 C3EF0A:08300:#08301:08302 <td:< td="">08303<td:< td="">08304<td:< td="">08304<td:< td="">08305<td:< td="">08306<td:< td="">08307<td:< td="">08308<td:< td="">08309<td:< td="">08309<td:< td="">08301<td:< td="">08302<td:< td="">08303<td:< td="">08303<td:< td="">08304<td:< td="">08305<td:< td="">08305<td:< td="">08311:08312<td:< td="">08313::08313::08314::08315<td::< td="">::08316<td::< td="">::::08311<td::< td="">:::::::::<</td::<></td::<></td::<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<></td:<>					;Update comparand pointer
OAEA C3E30A 08135 UMP Compare\$Dk\$Trk\$Loop 08137 ; ; raiables pointed at by HL to 08138 Hove\$Dk\$Trk; ; raiables pointed at by HL to 08139 i i variables pointed at by HL to 08139 Nove\$Dk\$Trk\$Loop: OB(14) NUI C,3 0AED 0E03 08141 MUV A,M ; Get source byte 0AF6 12 08144 STAX D ; Update pointers 0AF7 23 08145 INX D ; Update pointers 0AF2 23 08145 INX D ; Update pointers 0AF3 0D 08147 DCR C ; Count down on byte count 0AF3 0D 08147 DCR C ; Count down on byte count 0AF3 0D 08147 DCR C ; Count down on byte count 0AF3 0D 08147 DCR C ; Count down on byte count 0AF3 0D 08147 DCR ; Return if all bytes 08303 ; This routine moves eight bytes in a block; C times, from 08304 ; This routine		08133	DCR	С	;Count down on loop count
08136 ; 08138 Move\$Dk\$Trk: ; Moves the disk, track 08140 ; variables pointed at by DE 0AED 08140 ; those pointed at by DE 0AED 08141 MVI C.3 ; Disk (1), Track (2) 0AEF 7E 08142 Move\$Dk\$TrK\$Loop: Disk (1), Track (2) 0AF7 08144 STAX D ; Store in destination 0AF1 13 08145 INX D ; Uddate pointers 0AF2 08146 INX H ; Return if all bytes moved OAF3 (2) 0AF3 08146 INX H ; Return if all bytes moved OAF3 (2) 0AF3 08147 DCR ; Count down on byte count ; Return if all bytes moved 0AF3 08150 ;	0AE9 C8	08134	RZ		;Return (with zero flag set)
08137 i Move\$Dk%Trk: ; Variables pointed at by PL 0AED 0E03 08140 MVI C.3 ; Disk (1), Track (2) 0AED 0E03 08141 MVI C.3 ; Disk (1), Track (2) 0AEF 7E 08143 MOV A.M ; Get source byte 0AF0 12 08144 STAX D ; Update pointers 0AF1 13 08145 INX D ; Update pointers 0AF2 23 08146 INX H ; Count down on byte count 0AF3 0D 08147 DCR C ; Count down on byte count 0AF4 C8 08149 JMP Move\$Dk%Trk%Loop 0AF5 C3EF0A 08147 DCR C ; Count down on byte count 0AF3 0D 08147 DCR C ; Count down on byte count 0AF4 C8 08149 JMP Move\$Dk%Trk%Loop 0AF5 02F0A 08149 JMP Move\$Dk%Trk%Loop 0AF5 02 Move eight bytes ; ; 0AF3 05 (HL) to (DE). It uses "drop through" coding to speed ; 0AF3 05 (HL) to (DE). It uses "drop through" coding to speed	OAEA C3E30A	08135	JMP	Compare\$Dk\$Trk	\$Loop
ORISE (R139MoveSDk#Trk: (R140; Moves the disk, track (R140OAED 0E030814 (R141MVI C,3 (R142; those pointed at by DEOAED 7E08142 (R142MVX C,3 (R142; Disk (1), Track (2)OAEF 7E08142 (R142MOV A,M (R142; Store in destinationOAF1 1308145 (R142STAX D (R142; Store in destinationOAF2 2308146 (R142INX HOAF3 01 (AF4 C8 (R14208147 (R142DCR C (R142 (R142OAF5 C3EF0A (R14208147 (R142 (R142OAF5 C3EF0A (R14208147 (R142 (R142OAF5 C3EF0A (R142 <br< td=""><td></td><td>08136</td><td>;</td><td></td><td></td></br<>		08136	;		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$;		
OB140 i those pointed at by DE OAED 0E03 0B141 MUI C.3 ;Disk (1), Track (2) OAFF 12 0B143 MOV A.M. ;Get source byte OAF1 13 0B143 STAX D. ;Store in destination OAF1 13 0B144 INN D. ;Update pointers OAF2 23 0B146 INN H. OAF3 00 0B147 DCR C. OAF5 C3EFOA 0B148 RZ 0AF5 C3EFOA 0B149 JMP Move eight bytes ;Return if all bytes moved 0B300 ;# 0B303 ; 0B304 ; This routine moves eight bytes in a block, C times, from 0B305 ; (HL) to (DE). It uses "drop through" coding to speed 0B306 ; up execution. 0B307 ; DE -> destination address 0B310 ; C = number of 8-byte blocks to move 0B311 ; DE -> destination address 0B312 ; HL -> source address 0B312 ; HL -> source address 0B314 Move\$8			Move\$Dk\$Trk:		
OAED 0E03 08141 MVI C.3 fDisk (1), Track (2) OAEF 7E 08143 Move\$DistTrkElcop: fDisk (1), Track (2) OAF1 13 08144 STAX D fStore in destination OAF2 13 08145 STAX D fStore in destination OAF3 0D 08147 DCR fCurret for the second fCount down on byte count OAF3 0D 08147 DCR fCurret for the second fRturn if all bytes moved OAF5 CSEF0A 08148 RZ fRturn if all bytes moved fRturn if all bytes moved 08100 ; Move fight bytes g83001 ; g83002 ; 08303 ; This routine moves eight bytes in a block, C times, from g8303 ; 08305 ; (HL) to (DE). It uses "drop through" coding to speed g8307 ; 08307 ; C = number of 8-byte blocks to move g8311 ; DE -> destination address 08310 ; C = number of 8-byte from source G8313 ; 08310 ;					
OB142 Move\$DisTrk\$Lcop: OAEF 7E OB143 MOV A.M :Get source byte OAF0 12 OB144 STAX D :Store in destination OAF1 13 OB145 INX D :UPdate pointers OAF2 23 OB147 DCR C :Count down on byte count OAF3 0D OB147 DCR C :Count down on byte count OAF3 CB OB148 RZ :Return if all bytes moved OAF3 C3EFOA OB149 JMP Move\$Dik\$Trk\$Lcop OB303 :#		08140			; those pointed at by DE
OAEF OAF0 12OB143 08144MOV A,M;Get source byte istore in destinationOAF1 1308145INX INXDOAF2 2308145INX INXDOAF2 3008146INX INXHOAF3 0D08147DCR INXCOAF5 C3EF0A08148RZ INX;Return if all bytes movedOAF5 C3EF0A08149JMPMOVe\$Dk\$Trk\$Loop08300;#08300;#0830208303;Hove eight bytes08304;This routine moves eight bytes in a block, C times, from08305;(HL) to CDE). It uses "drop through" coding to speed08307;up execution.08308;Entry Parameters08309;C = number of 8-byte blocks to move08310;C = number of 8-byte from source0AF1 1308311;0AF2 1208316MOV0AF3 1308317;0AF4 1308317iNX0AF5 1208318MOV0AFF 1308317iNX0AFF 1308317iNX0AFF 1308317iNX0AFF 1408322iNX0AFF 15083220AFF 1608320STAX0AFF 1708221iNX0AFF 1808321iNX0AFF 1908321iNX0AFF 2008322iNX0AFF 2108323iNX0AFF 2208324STAX <t< td=""><td>OAED OEO3</td><td></td><td></td><td></td><td>;Disk (1), Track (2)</td></t<>	OAED OEO3				;Disk (1), Track (2)
OAF0 1208144STAXD:Store in destinationOAF1 1308145INXD:Update pointersOAF2 2308146INXHOAF3 0D08147DCRC:Count down on byte countOAF4 C808148RZ:Return if all bytes movedOAF5 C3EF0A08149JMPMove\$Dk\$Trk\$Loop08100;#		08142	Move\$Dk\$Trk\$Lo	op:	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$			MOV	Α,Μ	;Get source byte
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					
$\begin{array}{cccccccccccccccccccccccccccccccccccc$				-	;Update pointers
0AF4 C8 08148 R2 Return if all bytes moved 0AF5 C3EF0A 08149 JMP Move\$Dk\$Trk\$Loop 08100 ; 08300 ; 08301 ; 08302 ; Move eight bytes 08303 ; 08304 ; This routine moves eight bytes in a block, C times, from 08305 ; (HL) to (DE). It uses "drop through" coding to speed 08306 ; up execution. 08307 ; 08308 ; Entry Parameters 08309 ; 08310 ; C = number of 8-byte blocks to move 08311 ; DE \rightarrow destination address 08312 ; HL \rightarrow source address 08313 ; 08314 Move\$8: 08313 ; 08316 STAX D ;Put into destination 04F8 7E 08316 MOV A,M ;Get byte from source 04F8 13 08317 INX D ;Put into destination 04F2 12 08318 INX H 04FC 12 08320 STAX D ;Put into destination 04F2 13 08321 INX H 04FF 13 08321 INX H 04FF 13 08321 INX H 04FF 13 08321 INX H 04FF 12 08326 STAX D ;Put into destination 04FF 13 08321 INX H 04FF 13 08321 INX H 04FF 13 08321 INX H 0500 7E 08323 MOV A,M ;Get byte from source 04FF 13 08321 INX H 0500 7E 08324 STAX D ;Put into destination 04FF 13 08321 INX H 0500 7E 08323 MOV A,M ;Get byte from source 04FF 13 08321 INX H 0500 7E 08323 MOV A,M ;Get byte from source 04FF 13 08321 INX H 0500 7E 08323 MOV A,M ;Get byte from source 04FF 13 08321 INX H 0500 7E 08323 MOV A,M ;Get byte from source 04FF 13 08321 INX H 0500 7E 08323 MOV A,M ;Get byte from source 0501 12 08324 STAX D ;Put into destination 0502 13 08325 INX H 0503 7E 08329 INX H 0504 7E 08329 INX H 0506 7E 08329 INX H 0506 7E 08329 INX H 0506 7E 08329 INX H 0507 7E 08320 STAX D ;Put into destination 0506 12 08329 INX H 0508 7E 08331 MOV A,M ;Get byte from source 0507 12 08320 STAX D ;Put into destination 0506 13 08329 INX H 0508 7E 08331 MOV A,M ;Get byte from source 0507 23 08330 INX H					
$\begin{array}{cccccccccccccccccccccccccccccccccccc$				С	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	OAF4 C8				
08300 ;# 08301 ; 08302 ; Move eight bytes 08303 ; 08304 ; This routine moves eight bytes in a block, C times, from 08305 ; (HL) to (DE). It uses "drop through" coding to speed 08306 ; up execution. 08307 ; 08308 ; Entry Parameters 08309 ; 08310 ; C = number of 8-byte blocks to move 08311 ; DE -> destination address 08312 ; HL -> source address 08313 ; 04F8 7E 08315 MOV A,M ;Get byte from source 04F9 12 08316 STAX D ;Put into destination 04FA 13 08317 INX D ;Update pointers 04F2 08318 INX H 04FC 7E 08319 MOV A,M ;Get byte from source 04F1 12 08320 STAX D ;Put into destination 04FF 13 08321 INX B ;Update pointers 04FF 23 08322 INX H 0800 7E 08323 MOV A,M ;Get byte from source 0811 2 08324 STAX D ;Put into destination 04FF 23 08323 MOV A,M ;Get byte from source 0811 2 08324 STAX D ;Put into destination 0801 12 08324 STAX D ;Put into destination 0805 12 08328 STAX D ;Put into destination 0805 12 08328 STAX D ;Put into destination 0806 13 08329 INX H 0806 7E 08331 MOV A,M ;Get byte from source 0805 13 08329 INX H 0806 7E 08330 INX H 0806 7E 08331 MOV A,M ;Get byte from source 0805 12 08328 STAX D ;Put into destination 0806 13 08329 INX H 0807 72 08330 INX H 0808 7E 08331 MOV A,M ;Get byte from source	OAF5 C3EFOA			Move\$Dk\$Trk\$Lo	iop
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					
08302;Move eight bytes08303;This routine moves eight bytes in a block, C times, from08304;This routine moves eight bytes in a block, C times, from08305;(HL) to (DE). It uses "drop through" coding to speed08306;up execution.08307;up execution.08308;Entry Parameters08309;C = number of 8-byte blocks to move08310;C = number of 8-byte blocks to move08311;DE -> destination address08312;HL -> source address08313;;08314Move\$8:0AF8 7E08315MOV0AF8 1306317INX0AF2 1208318INX0AF2 1308317INX0AF2 1308321INX0AF2 1308322INX0AF2 1308321INX0AF2 1308322INX0B00 7E08323MOV0B1208324STAX0B03; Put into destination0AF5 2308324STAX0B00 7E08323MOV0B03 2308326INX0B03 2308327MOV0B04 7E08330INX0B05 7208330INX0B08 7E08331MOV0B08 7E08331MOV0B08 7E08331MOV0B08 7E08331MOV					
08303 ; 08304 ; This routine moves eight bytes in a block, C times, from 08305 ; (HL) to (DE). It uses "drop through" coding to speed 08306 ; up execution. 08307 ; 08308 ; Entry Parameters 08309 ; 08310 ; C = number of 8-byte blocks to move 08311 ; DE -> destination address 08312 ; HL -> source address 08313 ; 08314 Move\$3: 04F8 7E 08315 MOV A,M ;Get byte from source 04F9 12 08316 STAX D ;Put into destination 04F4 13 08317 INX D ;Put into destination 04F2 04518 INX H 04FC 7E 08319 MOV A,M ;Get byte from source 04F9 12 08318 INX H 04FC 7E 08319 MOV A,M ;Get byte from source 04F1 12 08320 STAX D ;Put into destination 04FE 13 08321 INX D ;Put and the stination 04FE 13 08321 INX D ;Put and the stination 0800 7E 08323 MOV A,M ;Get byte from source 0801 12 08324 STAX D ;Put into destination 0802 13 08325 INX D ;Put into destination 0803 10 08320 STAX D ;Put into destination 0804 7E 08327 MOV A,M ;Get byte from source 0805 12 08328 STAX D ;Put into destination 0806 13 08329 INX H 0808 7E 08331 MOV A,M ;Get byte from source 0807 23 08330 INX H				icht huter	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$				Iduit Dytes	
08305 ; (HL) to (DE). It uses "drop through" coding to speed 08306 ; up execution. 08307 ; 08308 ; Entry Parameters 08309 ; 08310 ; C = number of 8-byte blocks to move 08311 ; DE -> destination address 08312 ; HL -> source address 08313 ; 08314 Move\$8: 08457 E 08315 MOV A,M ;Get byte from source 04F8 7E 08316 STAX D ;Put into destination 04F8 12 08316 STAX D ;Put into destination 04F8 13 08317 INX D ;Update pointers 04F9 12 08318 INX H 04FC 7E 08319 MOV A,M ;Get byte from source 04F9 12 08320 STAX D ;Put into destination 04FE 13 08321 INX D ;Update pointers 04FF 23 09322 INX H 0800 7E 08323 MOV A,M ;Get byte from source 04FF 12 08326 INX H 0800 7E 08323 MOV A,M ;Get byte from source 0801 12 08324 STAX D ;Put into destination 0802 13 08325 INX D ;Update pointers 0804 7E 08327 MOV A,M ;Get byte from source 0805 12 08328 STAX D ;Update pointers 0806 7E 08327 MOV A,M ;Get byte from source 0806 13 08329 INX H 0806 13 08329 INX H 0806 13 08329 INX H 0807 20 08330 INX H				outine moves ein	ht hytes in a block. C times from
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					arop through courny to speed
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					
$\begin{array}{cccccccccccccccccccccccccccccccccccc$			•	Parameters	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					
$\begin{array}{cccccccccccccccccccccccccccccccccccc$				C = number of	8-byte blocks to move
$\begin{array}{cccccccccccccccccccccccccccccccccccc$					
08313;08314Move\$8:0AF8 7E08315MOVA,M0AF9 1208316STAXD0AF2 1308317INXD0AF2 1308317INXD0AF2 1208318INXH0AF2 1308317INXD0AF2 1408317INXH0AF2 7E08319MOVA,M0AF2 1308321INXH0AF2 1308321INXD0AF2 1308322INXH0AF2 1308322INXH0B00 7E08323MOVA,M0B01 1208324STAXD0B03 2308326INXH0B04 7E08327MOVA,M0B05 1208328STAXD0B06 1308329INXP0B06 1308329INXH0B08 7E08331MOVA,M0B08 7E08331MOVA,M1000 7208330INXH			•		
OB314 Move\$8: OAF8 7E 08315 MOV A, M ; Get byte from source OAF9 12 08316 STAX D ; Put into destination OAFA 13 08317 INX D ; Put into destination OAF2 32 08318 INX H OAFC 7E 0830 STAX D ; Put into destination OAFE 13 08321 INX H OAFE 23 08322 INX D ; Put into destination OAFE 13 08321 INX D ; Put into destination OAFE 23 08322 INX H 0B00 7E 08323 MOV A, M ; Put into destination 0B01 12 08324 STAX D ; Put into destination 0B03 23 08325 INX D ; Put into destination 0B03 7E 08327 MOV A, M ; Get byte from source 0B06 13 08329 INX D ; Put into destination			;		
OAF8 7EOB315MOVA,M;Get byte from sourceOAF9 12OB316STAXD;Put into destinationOAFA 13OB317INXD;Update pointersOAFB 23OB318INXHOAFC 7EOB319MOVA,M;Get byte from sourceOAF2 12OB320STAXD;Put into destinationOAFE 13OB321INXD;Put into destinationOAFE 23OB322INXHOB00 7EOB323MOVA,M;Get byte from sourceOB01 12OB324STAXD;Put into destinationOB02 13OB325INXHOB03 23OB326INXHOB04 7EOB327MOVA,M;Get byte from sourceOB05 12OB328STAXD;Put into destinationOB06 13OB329INXD;Update pointersOB06 723OB330INXHOB08 7EOB331MOVA,M;Get byte from source			Move\$8:		
OAF9 12 OB316 STAX D ;Put into destination OAFA 13 OB317 INX D ;Update pointers OAFE 23 OB318 INX H OAFC 7E OB319 MOV A,M ;Get byte from source OAFE 13 OB320 STAX D ;Update pointers OAFE 13 OB321 INX H	OAF8 7E		MOV	Α,Μ	;Get byte from source
OAFA 13O8317INXD;Update pointersOAFB 2308318INXHOAFC 7E08319MOVA,M;Get byte from sourceOAFD 1208320STAXD;Put into destinationOAFE 1308321INXHOBO 7E08322INXHOBO 7E08323MOVA,M;Get byte from sourceOBO 7E08324STAXD;Put into destinationOBO 7E08325INXHOBO3 2308326INXHOBO4 7E08327MOVA,M;Get byte from sourceOBO5 1208328STAXD;Put into destinationOBO6 1308329INXH;Get byte from sourceOBO7 2308330INXH;Update pointersOBO8 7E08331MOVA,M;Get byte from source	0AF9 12		STAX		
OAFC 7E OB319 MOV A,M ;Get byte from source OAFD 12 OB320 STAX D ;Put into destination OAFE 13 OB321 INX D ;Update pointers OAFE 23 OB322 INX H	OAFA 13	08317	INX	D	
OAFD 12 O8320 STAX D ;Put into destination OAFE 13 O8321 INX D ;Update pointers OAFE 23 O8322 INX H OB00 7E O8323 MOV A,M ;Get byte from source OB01 12 O8324 STAX D ;Put into destination OB02 13 O8325 INX D ;Put into destination OB03 23 O8326 INX H OB04 7E O8327 MOV A,M ;Get byte from source OB05 12 O8328 STAX D ;Put into destination OB06 13 O8329 INX D ;Update pointers OB07 23 O8330 INX H OB08 7E O8331 MOV A,M ;Get byte from source	OAFB 23	08318	INX	н	
OAFE 13 O8321 INX D ;Update pointers OAFF 23 08322 INX H OBO0 7E 08323 MOV A, M ;Get byte from source 0B00 7E 08323 MOV A, M ;Get byte from source 0B01 12 08324 STAX D ;Put into destination 0B03 08326 INX H 0B04 7E 08327 MOV A, M ;Get byte from source 0B05 13 08329 STAX D ;Put into destination 0B06 13 08329 INX D ;Update pointers 0B07 23 08330 INX H 0B08 7E 08331 MOV A, M ;Get byte from source	OAFC 7E	08319	MOV	Α,Μ	;Get byte from source
OAFF 23 08322 INX H OBO0 7E 08322 INX H OBO0 7E 08322 MOV A,M ;Get byte from source OBO1 12 08324 STAX D ;Put into destination OBO2 13 08325 INX D ;Update pointers OBO3 23 08326 INX H OBO4 7E 08327 MOV A,M ;Get byte from source OBO5 12 08328 STAX D ;Put into destination OBO6 13 08329 INX D ;Update pointers OBO7 23 08330 INX H OBO8 7E 08331 OBO8 7E 08331 MOV A,M ;Get byte from source	OAFD 12	08320	STAX		;Put into destination
OB00 7E OB323 MOV A,M ;Get byte from source OB01 12 OB324 STAX D ;Put into destination OB02 13 OB325 INX D ;Update pointers OB03 23 OB326 INX H OB04 7E OB327 MOV A,M ;Get byte from source OB05 12 OB328 STAX D ;Put into destination OB06 13 OB329 INX D ;Update pointers OB07 23 OB330 INX H OB08 7E OB331 MOV A,M ;Get byte from source					;Update pointers
OB01 12 OB324 STAX D ;Put into destination OB02 13 OB325 INX D ;Update pointers OB03 OB326 INX H OB04 7E OB327 MOV A,M ;Get byte from source OB05 13 OB329 INX D ;Put into destination OB06 13 OB329 INX D ;Put into destination OB07 23 OB330 INX H OB08 7E OB331 MOV A,M ;Get byte from source					
0802 13 08325 INX D ;Update pointers 0803 23 08326 INX H 0804 7E 08327 MOV A,M ;Get byte from source 0805 12 08328 STAX D ;Put into destination 0806 13 08329 INX D ;Update pointers 0807 23 08330 INX H 0 0808 7E 08331 MOV A,M ;Get byte from source					
0B03 23 0B326 INX H 0B04 7E 0B327 MOV A,M ; Get byte from source 0B05 12 0B328 STAX D ; Put into destination 0B06 13 0B329 INX D ; Update pointers 0B07 23 0B330 INX H 0B08 7E 0B331 MOV A,M ; Get byte from source					
OB04 7E O8327 MOV A,M ;Get byte from source OB05 12 O8328 STAX D ;Put into destination OB06 13 O8329 INX D ;Update pointers OB07 23 O8330 INX H OB08 7E 08331 MOV A,M ;Get byte from source					;Update pointers
OB05 12 OB328 STAX D ; Put into destination OB06 13 OB329 INX D ; Update pointers OB07 23 OB330 INX H OB08 7E OB331 MOV A,M ; Get byte from source	OB03 23				
0806 13 08329 INX D ;Update pointers 0807 23 08330 INX H 0808 7E 08331 MOV A,M ;Get byte from source					
0807 23 08330 INX H 0808 7E 08331 MOV A,M ;Get byte from source					
0B08 7E 08331 MOV A,M ;Get byte from source					;update pointers
					.Cat but from course
	0507 12	08332	514%	U	Frat THEO DESCENDENCE

OBOA 13	08333	INX	D	+Undat	te pointers
OBOB 23	08334		ท ี่	, opuar	e poincers
OBOC 7E	08335		A, M	+Get h	byte from source
OBOD 12	08336		D		into destination
OBOE 13	08337		Ď		te pointers
OBOF 23	08338		ĥ	,0904(e pointers .
0B10 7E	08339		 А, М	+Gat b	byte from source
OB11 12	08340		D		into destination
OB12 13	08341		D		te pointers
OB13 23	08342		ค ้	,0000	
0B14 7E	08343		A, M	:Get b	byte from source
OB15 12	08344		D		into destination
OB16 13	08345		- D		te pointers
0B17 23	08346		Ĥ	, op dat	
0017 20	08347	1144			
OB18 OD	08348	DCR	C.	:Count	down on loop counter
0B19 C2F80A	08349		Move\$8		at until done
OBIC C9	08350	RET		,	
	08351				
	08352				
	08500	;#			
	08501	;			
	08502		tion to the	disk contr	ollers on this computer system⊾
	08503	;			
	08504		e two "smart	" disk con	ntrollers on this system, one
	08505	; for the	8" floppy di	skette dri	ives, and one for the 5 1/4"
	08506		kette drives		
	08507	;		-	
	08508		rollers are	"hard-wire	d" to monitor certain locations
	08509	: in memor	y to detect	when they	are to perform some disk
	08510	; operatio	n. The 8" co	ntroller 1	looks at location 0040H, and
	08511	the 5 1/	4" controlle	r looks at	location 0045H. These are
	08512				es. If the most significant
	08513				set, the controller will then
	08514	; look at	the word fol	lowing the	e respective control bytes.
	08515	: This wor	d must conta	in the add	tress of a valid disk control
	08516	; table th	at specifies	the exact	t disk operation to be performed.
	08517	;			
	08518				ompleted, the controller resets
	08519	; its disk	control byt	e to OOH,	and this indicates completion
	08520	; to the d	isk driver c	ode.	
	08521	;			
	08522				turn code in a disk status block.
	08523	; Both.com	trollers use	the same	location (0043H) for this.
	08524	; If the f	irst byte of	this stat	tus block is less than 80H, then
	08525	; adiske	rror has occ	urred. For	r this simple BIOS, no further details
	08526				levant. Note that the disk controller
	08527	; has buil	t-in retry l	ogic, read	s and writes are attempted ten
	08528	; times be	fore the con	troller re	eturns an error.
	08529	;			
i	08530				is shown below. Note that the
	08531				ty for control tables to be
1	08532				quence of disk operations can
1	08533				is feature is not used. However,
1	08534				the chain pointers in the
	08535				d back to the main control bytes
1	08536	; in order	to indicate	the end o	of the chain.
	08537	1			
0040 =	08538	Disk\$Control\$8	EQU	40H	;8" control byte
0041 =	08539	Command\$Block\$8	EQU	41H	;Control table pointer
	08540	1			
0043 =	08541	Disk\$Status\$Bloc	k EQU	43H	;8" AND 5 1/4" status block
	08542	1		45H	- E 1/48
0045 =	08543	Disk\$Control\$5	EQU		;5 1/4" control byte
0046 =	08544	Command\$Block\$5	EQU	46H	;Control table pointer
	08545	,			
	08546	; Eleppy I	hel Contart	Tables	
	08547	; Floppy I	lisk Control	190162	
0010 00	08548	7 Elenny#0		DB	0 :Command
OB1D OO	08549	Floppy\$Command:		EQU	0 ycommand 01H
0001 =	08550	Floppy\$Read\$Code		EQU	02H
0002 =	08551 08552	Floppy\$Write\$Cod Floppy\$Unit:	ie	DB	02H 0 ;Unit (drive) number = 0 or 1
OBIE 00					
0B1F 00	08553	Floppy\$Head:		DB	0 ;Head number = 0 or 1
0B20 00	08554	Floppy\$Track:		DB	0 ;Track number 0 :Sector number
0B21 00	08555	Floppy\$Sector:		DB	0 ;Sector number
L					

0B22 0000	08556	Floppy\$Byte\$Cou	int:	DW	0	;Number of bytes to read/write
0B24 0000	08557	Floppy\$DMA\$Addr		DW	õ	;Transfer address
0B26 0000	08558	Floppy\$Next\$Sta		DW	ŏ	Pointer to next status block
0020 0000	08559	110000000000000000000000000000000000000	itas#biock:	2.44	v	; if commands are chained.
0B28 0000	08560	Floppy\$Next\$Cor	trols costion.	DW	0	Pointer to next control byte
0520 0000	08561	110000000000000000000000000000000000000	thor proceedion.	DW	v	; if commands are chained
	08562	;				, if commands are chained
	08700	, ;#				
	08701	; "				
	08702	-				
	08702	, Write\$No\$Debloo		·Urito	content	ts of disk buffer to
	08704	WITCEPHOPDEBIOC	K.		ect sec	
0B2A 3E02	08705	MVI	A,Floppy\$Writes			write function code
0B2C C3310B	08706	JMP	Common\$No\$Deblo			o common code
0620 033106	08707	Read\$No\$Deblock				sly selected sector
	08708	Keau #NO#DeDIOCK	•		disk b	
0B2F 3E01	08709	MVI	A,Floppy\$Read\$0			read function code
ODEN OLOY	08710	Common\$No\$Deblc	ck:		,	
0B31 321D0B	08711	STA	Floppy\$Command	•Set co	mmand f	Function code
ODDI OZIDOD	08712	0111	110000000000000000000000000000000000000			blocked command table
	08713			,		
0B34 3A360A	08714	LDA	Selected\$Disk\$]	TVDe	:Check	<pre>< if memory disk operation</pre>
0B37 FE03	08715	CPI	M\$Disk	.,	,	(if memory with open direct
0B39 CA7A0B	08716	JZ	M\$Disk\$Transfer	Yes, i	t is Ma	BDisk
	08717					
	08718	No\$Deblock\$Retr	y:	:Re-ent	ry poir	nt to retry after error
0B3C 218000	08719	LXI	H. 128		per sec	
0B3F 22220B	08720	SHLD	Floppy\$Byte\$Cou			
OB42 AF	08721	XRA	Α		ppy onl	ly has head O
0B43 321F0B	08722	STA	Floppy\$Head			
	08723			;		
OB46 3A2DOA	08724	LDA	Selected\$Disk	;8" flo	орру сог	ntroller only knows about
	08725					d 1 so Selected\$Disk must
	08726			; be c	converte	e di
0B49 E601	08727	ANI	01H	;Turn i	into O c	or 1
OB4B 321EOB	08728	STA	Floppy\$Unit	;Set ur	nit numb	ber
	08729			;		
OB4E 3A2E0A	08730	LDA	Selected\$Track			
0B51 32200B	08731	STA	Floppy\$Track	;Set tr	ack num	nber
	08732			;		
0B54 3A300A	08733	LDA	Selected\$Sector			
0B57 32210B	08734	STA	Floppy\$Sector	;Set se	ector nu	umber
	08735			;		
0B5A 2AA609	08736	LHLD	DMA\$Address	; Iransi	er dire	ectly between DMA Address
0B5D 22240B	08737	SHLD	Floppy\$DMA\$Add	ress	; and	8" controller.
	08738			· • • • • • •		troller can accept chained
	08739 08740					ol tables, but in this case,
	08740			; 0157		ot used, so the "Next" pointers
	08742					inted back at the initial
	08742					tes in the base page.
0B60 214300	08743	LXI	H,Disk\$Status\$			Point next status back at
0B60 214300 0B63 22260B	08745	SHLD	Floppy\$Next\$Sta		· k	; main status block
ODOD ZZZOUD	08746	Cine D				t main status stock
0B66 214000	08747	LXI	H,Disk\$Control	88		, Point next control byte
0B69 22280B	08748	SHLD	Floppy\$Next\$Con		ation	; back at main control byte
VEO/ ELECOD	08749				;	
0B6C 211D0B	08750	LXI	H,Floppy\$Comman	nd	Point	t controller at control table
0B6F 224100	08751	SHLD	Command\$Block\$			
	08752				;	
0B72 214000	08753	LXI	H,Disk\$Control	\$8	Activ	vate controller to perform
0B75 3680	08754	MVI	M, 80H			eration
0B77 C33B0C	08755	JMP	Wait\$For\$Disk\$(Complete		
	08756					
	08757	;				
	08900	; #				
	08901	; Memory	disk driver			
	08902	;				
	08903	; This ro	outine must use a	an intern	nediary	buffer, since the
	08904		lress in bank (""	track") () occup	ies the same
	08905	; place i	n the overall ac	dress sp	ace as	the M\$Disk itself.
	08906	; The M\$D	lisk\$Buffer is at	oove the	48K mar	rk, and therefore
	08907	; remains	in the address	space re	gardles	ss of which bank/track
	08908	; is sele	cted.			
	08909	;				
	08910	;				

08911 For writing, the 128-byte sector must be processed: ; 08912 08913 1. Move sector DMA\$Address -> M\$Disk\$Buffer Select correct track (+1 to get bank number)
 Move sector M\$Disk\$Buffer -> M\$Disk image 08914 08915 08916 4. Select bank 0 08917 08918 For reading, the processing is: . 08919 08920 1. Select correct track/bank / 08921 2. Move sector M\$Disk image -> M\$Disk\$Buffer 08922 3. Select Bank O 08923 4. Move sector M\$Disk\$Buffer -> DMA\$Address 08924 08925 If there is any risk of any interrupt causing control to be transferred to an address below 48K, interrupts must be disabled when any bank other than 0 is selected. 08926 08927 08928 M\$Disk\$Transfer: 08929 0B7A 3A300A I DA Selected\$Sector ;Compute address in memory 08930 ; by muliplying sector * 128 OB7D 6F 08931 MOV L.A **OB7E 2600** 08932 MVI н, о 0B80 29 08933 DAD н ;* 2 0B81 29 08934 DAD ;* 4 н OB82 29 08935 DAD н ;* 8 0B83 29 08936 DAD н ;* 16 OB84 29 08937 DAD н ;* 32 ;* 64 0B85 29 08938 **DAD** н DAD :* 128 OB86 29 08939 н 08940 Selected\$Track ;Compute which half of bank sector ; is in by using LS bit of track B,A ;Save copy for later 0B87 3A2E0A 08941 I DA 08942 **OB8A 47** 08943 MOV ;Isolate lower/upper indicator **OB8B E601** 08944 ANI OBSD CA940B M\$Disk\$Lower\$Half 08945 JZ 08946 0B90 110060 0B93 19 08947 LXI $D_{1}(48 \times 1024) / 2$;Upper half, so bias address 08948 DAD n 08949 ;HL -> sector in memory M\$Disk\$Lower\$Half: 08950 A.B Recover selected track **0B94** 78 08951 MOV ;Divide by 2 to get bank number ;Bank 1 is first track 0895 1F 08952 RAR 0B96 3C 08953 INR Α 0B97 47 08954 MOV B.A Preserve for later use 08955 OB98 3A1D0B 08956 LDA Floppy\$Command ;Check if reading or writing 0B9B FE02 CPI Floppy\$Write\$Code 08957 ;Writing OB9D CABEOB 08958 JZ M\$Disk\$Write 08959 ;Reading 08960 Select\$Bank OBAO CDDDOB 08961 CALL ;Select correct memory bank OBA3 113023 OBA6 0E10 08962 LXI D,M\$Disk\$Buffer ;DE -> M\$Disk\$Buffer, HL -> M\$Disk image 08963 MVI C,128/8 ;Number of 8-byte blocks to move OBA8 CDF80A 08964 CALL Move\$8 08965 MVI в,о **OBAB 0600** 08966 :Revert to normal memory bank OBAD CDDDOB 08967 CALL Select\$Bank 08968 08969 LHLD DMA\$Address ;Get user's DMA address OBB0 2AA609 OBB3 113023 08970 LXI D,M\$Disk\$Buffer ;DE -> User's DMA, HL -> M\$Disk buffer ;Number of 8-byte blocks to move OBB6 EB 08971 XCHG OBB7 OE10 08972 MVI C.128/8 OBB9 CDF80A 08973 CALL Move\$8 08974 08975 OBBC AF XRA Α :Indicate no error OBBD C9 08976 RFT 08977 08978 M\$Disk\$Write: ;Writing OBBE E5 08979 PUSH Save sector's address in M\$Disk image н DMA\$Address OBBF 2AA609 08980 LHLD ;Move sector into M\$Disk\$Buffer OBC2 113023 08981 LXI D,M\$Disk\$Buffer 0BC5 0E10 08982 MVI C,128/8 ;Number of 8-byte blocks to move ;(Does not use B register) ;B = memory bank to select OBC7 CDF80A 08983 CALL Move\$8 00004 OBCA CDDDOB CALL Select\$Bank 08985 08986

Figure 8-10. (Continued)

OBCD		08987	POP	D	<pre>#Recover sector's M\$Disk image address</pre>
	213023	08988	LXI	H,M\$Disk\$Buffer	
OBD1		08989	MVI	C,128/8	
OBD3	CDF80A	08990 08991	CALL	Move\$8	;Move into M\$Disk image
OBD6	0/00		MVI	B.0	
		08992			;Select bank O
0808	CDDDOB	08993	CALL	Select\$Bank	
		08994		•	
OBDB		08995	XRA	Α	;Indicate no error
OBDC	69	08996 08997	RET		
		09100	;		
		09100	;# : Select	h and	
				Dank	
		09102 09103	; This u		- Aba waanidwad waxann baali
		09103	; Thisr	bat the bandward	n the required memory bank. port that controls bank selection
		09105			it. These are preserved across
		09106		elections.	It. mese are preserved across
		09107	; banks ;	elections.	
		09108		parameter	
		09109		parameter	
		09110	7		
		09111	1	B ≕ bank number	
0040	-	09112	; Bank\$Control\$P	ort EQU	4 0H
0040 00F8		09112	Bank%Lontrol%P Bank%Mask	EQU	1111\$1000B ;To preserve other bits
00-8	-	09113		EWO	strate other bits
		09115	, Select\$Bank:		
OBDD	DRAO	09116	IN	Bank\$Control\$Pc	rt ;Get current setting in port
OBDF		09117	ANI	Bank\$Mask	Preserve all other bits
OBE 1		09118	ORA	B	;Set bank code
OBE2		09119	OUT	Bank\$Control\$Pc	
OBE4		09120	RET	Dank #Control #	, Select the bank
ODC4	07	09121			
		09200	; #		
		09201	1		
		09202	•		
		09203	Write\$Physical		;Write contents of disk buffer to
		09204	Willeringsical	•	; correct sector
OBE5	3E02	09205	MVI	A,Floppy\$Write\$	
	C3ECOB	09206	JMP		;Go to common code
		09206 09207			;Go to common code ;Read previously selected sector
OBE7	C3ECOB	09206 09207 09208	JMP Read\$Physical:	Common\$Physical	;Go to common code ;Read previously selected sector ; into disk buffer
OBE7	C3ECOB	09206 09207 09208 09209	JMP		;Go to common code ;Read previously selected sector ; into disk buffer
OBE7	C3ECOB	09206 09207 09208 09209 09210	JMP Read\$Physical: MVI ;	Common\$Physical A,Floppy\$Read\$C	;Go to common code ;Read previously selected sector ; into disk buffer
OBE7	C3ECOB 3E01	09206 09207 09208 09209 09210 09211	JMP Read\$Physical: MVI ; Common\$Physica	Common\$Physical A,Floppy\$Read\$C 1:	;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code
OBE7	C3ECOB	09206 09207 09208 09209 09210 09211 09212	JMP Read\$Physical: MVI ;	Common\$Physical A,Floppy\$Read\$C 1:	;Go to common code ;Read previously selected sector ; into disk buffer
OBE7	C3ECOB 3E01	09206 09207 09208 09209 09210 09211	JMP Read\$Physical: MVI ; Common\$Physica	Common\$Physical A,Floppy\$Read\$C 1:	;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code
OBE7	C3ECOB 3E01	09206 09207 09208 09209 09210 09211 09212 09213 09214	JMP Read\$Physical: MVI ; Common\$Physica STA ;	Common\$Physical A,Floppy\$Read\$C 1:	;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table
OBE7 OBEA OBEC	C3ECOB 3E01	09206 09207 09208 09209 09210 09211 09212 09213	JMP Read\$Physical: MVI ; Common\$Physica	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command	;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error
OBE7 OBEA OBEC OBEF	C3EC0B 3E01 321D0B 3A2A0A	09206 09207 09208 09209 09210 09211 09212 09213 09214 09215 09216	JMP Read\$Physical: ////////////////////////////////////	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer
OBE7 OBEA OBEC OBEF OBF2	C3ECOB 3E01 321D0B 3A2A0A FE01	09206 09207 09208 09209 09210 09211 09212 09213 09214 09215	JMP Read\$Physical: , Common\$Physica STA ; Deblock\$Retry: LDA CPI	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy</pre>
OBE7 OBEA OBEC OBEF OBF2 OBF4	C3ECOB 3E01 321D0B 3A2A0A FE01 CAFD0B	09206 09207 09208 09209 09210 09211 09212 09213 09214 09215 09216 09217	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes</pre>
OBE7 OBEA OBEC OBEF OBF2 OBF4 OBF7	C3ECOB 3E01 321D0B 3A2A0A FE01 CAFD0B	09206 09207 09208 09209 09210 09211 09212 09213 09214 09215 09216 09217 09218	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error</pre>
OBE7 OBEA OBEC OBEF OBF2 OBF4 OBF7 OBF9	C3ECOB 3E01 321D0B 3A2A0A FE01 CAFD0B 3E01 32320A	09206 09207 09208 09209 09211 09212 09213 09214 09215 09216 09217 09218 09219 09220	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error</pre>
OBE7 OBEA OBEC OBEF OBF2 OBF4 OBF7 OBF9	C3ECOB 3E01 321D0B 3A2A0A FE01 CAFD0B 3E01 32320A	09206 09207 09208 09209 09211 09212 09213 09214 09215 09216 09217 09218 09219	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ JZ MVI STA RET	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error</pre>
OBE7 OBEA OBEC OBEF OBF2 OBF4 OBF7 OBF9	C3ECOB 3E01 321D0B 3A2A0A FE01 CAFD0B 3E01 32320A	09206 09207 09208 09209 09210 09212 09212 09214 09215 09216 09217 09218 09219 09220 09221	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error</pre>
OBE7 OBEA OBEC OBF2 OBF2 OBF4 OBF7 OBF9 OBFC	C3ECOB 3E01 321D0B 3A2A0A FE01 CAFD0B 3E01 32320A	09206 09207 09208 09209 09210 09212 09213 09214 09215 09216 09216 09218 09219 09218	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ JZ MVI STA RET	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ;</pre>
OBE7 OBEA OBEC OBF2 OBF4 OBF7 OBF7 OBFC OBFD	C3ECOB 3E01 321DOB 3A2A0A FE01 CAFD0B 3E01 32320A C9 3A270A	09206 09207 09208 09209 09211 09212 09213 09214 09215 09215 09215 09215 09217 09218 09219 09220 09220 09222 09222 09223	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1</pre>
OBE7 OBEA OBEC OBF2 OBF4 OBF7 OBF9 OBFC OBFD OCOO	C3ECOB 3E01 321DOB 3A2A0A FE01 CAFD0B 3E01 32320A C9 3A270A	09206 09207 09208 09209 09210 09211 09212 09213 09214 09215 09216 09216 09217 09218 09219 09220 09221	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ JZ MVI STA RET Correct\$Disk\$T	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ;</pre>
OBE7 OBEA OBEC OBF2 OBF4 OBF7 OBF9 OBFC OBFD OCOO	C3EC0B 3E01 321D0B 3A2A0A FE01 CAFD0B 3E01 32320A C9 3A270A E601	09206 09207 09208 09209 09211 09212 09213 09214 09215 09216 09216 09217 09218 09219 09221 09221 09221 09221 09221 09222 09223	JMP Read\$Physical: ////////////////////////////////////	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1</pre>
OBE7 OBEA OBEC OBF2 OBF4 OBF7 OBF7 OBF7 OBF7 OBFC	C3EC0B 3E01 321D0B 3A2A0A FE01 CAFD0B 3E01 32320A C9 3A270A E601 321E0B	09206 09207 09208 09209 09211 09211 09212 09213 09214 09215 09216 09217 09220 09220 09222 09222 09223 09224 09225 09224	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1 ; for disk controller</pre>
0BE7 0BEA 0BEC 0BF2 0BF4 0BF7 0BF7 0BF7 0BF7 0BFC 00F0 00F0 00F0 0000 00002	C3EC0B 3E01 321D0B 3A2A0A FCAFD0B 3E01 320A C9 3A270A E601 321E0B 2A280A	09206 09207 09208 09209 09210 09211 09212 09213 09214 09215 09216 09217 09218 09217 09228 09221 09222 09222 09222 09222 09222 09222 09224 09225	JMP Read\$Physical: ////////////////////////////////////	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command Correct\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1 ; for disk controller ;Set up head and track number</pre>
0BE7 0BEA 0BEC 0BF2 0BF2 0BF7 0BF9 0BF7 0BF9 0BFC 0BFD 0C00 0C02 0C05 0C08	C3EC0B 3E01 321D0B 3A2A0A FE01 32320A C9 3A270A E601 321E0B 2A280A 70	09206 09207 09208 09209 09211 09211 09212 09213 09214 09215 09216 09217 09220 09220 09222 09222 09223 09224 09225 09224	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA RET Correct\$Disk\$T LDA ANI STA LHLD	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; for disk control table ; for disk controller ;Set up head and track number ;Even numbered tracks will be on</pre>
0BE7 0BEA 0BEC 0BF2 0BF2 0BF4 0BF9 0BF7 0BFC 0BFD 0C02 0C02 0C05 0C09	C3EC0B 3E01 321D0B 342A0A FE01 CAFD0B 3E01 32320A C9 3A270A E401 321E0B 2A280A 7D E401	09206 09207 09208 09209 09210 09212 09212 09214 09215 09216 09217 09218 09217 09220 09221 09222 09223 09225 09225 09225 09225 09225 09225	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA LDA ANI STA	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L 1	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1 ; for disk control table ; ;Convert disk number to 0 or 1 ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered on head 1</pre>
0BE7 0BEA 0BEC 0BF2 0BF2 0BF4 0BF9 0BF7 0BFC 0BFD 0C02 0C02 0C05 0C09	C3EC0B 3E01 321D0B 3A2A0A FE01 32320A C9 3A270A E601 321E0B 2A280A 70	09206 09207 09208 09209 09211 09212 09213 09214 09215 09216 09216 09217 09218 09219 09221 09221 09222 09221 09222 09223 09224 09225 09226 09227 09228 09229	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA ANI STA	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; for disk control table ; for disk controller ;Set up head and track number ;Even numbered tracks will be on</pre>
0BE7 0BEA 0BEC 0BF2 0BF2 0BF7 0BF9 0BFC 0BF0 00BF0 00C02 0C05 0C05 0C05 0C08	C3EC0B 3E01 321D0B 3A2A0A FE01 CAFD0B 3E01 32320A C9 3A270A E601 321E0B 2A280A 7D E601 321F0B	09206 09207 09208 09209 09211 09212 09213 09214 09215 09215 09215 09215 09217 09220 09221 09222 09223 09224 09225 09227 09228 09227 09228 09229 09230	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA LDA ANI STA	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L Floppy\$Head	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; for disk control table ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered on head 1 ;Set head number</pre>
08E7 08EA 08EC 08F2 08F4 08F7 08F7 08F7 08F7 08F7 008F0 00C00 00C00 00C00 00C00 00C08 00C08 00C08	C3EC0B 3E01 321D0B 3A2A0A FCATD0B 3E01 3201 320A C9 3A270A 5601 321E0B 2A280A 7D 2A280A 7D 2A280A 7D	09206 09207 09208 09209 09211 09212 09213 09214 09214 09215 09216 09217 09218 09217 09221 09220 09221 09222 09223 09224 09225 09226 09227 09228 09227 09228 09229 09229 09223	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA RET Correct\$Disk\$T LDA ANI STA LHLD MOV ANI	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L 1	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1 ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered on head 1 ;Set head number ;Note: this is single byte value</pre>
08E7 08EA 08EC 08F2 08F2 08F3 08F9 08F7 08F9 08F7 08F9 08F0 0060 00002 0005 0008 0009 00008	C3EC0B 3E01 321D0B 3A2A0A FE01 3CAFD0B 3E01 32320A C9 3A270A E601 321E0B 2A280A 7D E401 321F0B 7D F	09206 09207 09209 09219 09211 09212 09213 09214 09215 09215 09215 09216 09217 09227 09220 09222 09222 09222 09224 09225 09224 09225 09228 09229 09230 09231 09232 09232	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA ANI STA ANI STA	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L Floppy\$Head	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; for disk control table ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered on head 1 ;Set head number</pre>
08E7 08EA 08EC 08EF2 08F7 08F7 08F9 08F7 08F9 08F0 08F0 0007 0000 00002	C3EC0B 3E01 321D0B 3A2A0A FCATD0B 3E01 3201 320A C9 3A270A 5601 321E0B 2A280A 7D 2A280A 7D 2A280A 7D	09206 09207 09208 09209 09211 09212 09213 09214 09215 09216 09217 09220 09223 09223 09224 09222 09223 09224 09225	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA RET Correct\$Disk\$T LDA ANI STA LHLD MOV ANI	Common\$Physical A,Floppy\$Read\$C I: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L 1 Floppy\$Head A,L	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1 ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered on head 1 ;Set head number ;Note: this is single byte value</pre>
08E7 08EA 08EC 08F2 08F2 08F7 08F9 08F7 08F9 008F7 008F0 0067 0000 00002 00005 00008 00008 00008	C3EC0B 3E01 321D0B 3A2A0A FE01 32320A C9 3A270A 5601 321E0B 2A280A 7D E601 321F0B 7D 1F 32200B	09206 09207 09208 09209 09211 09212 09213 09214 09214 09215 09216 09217 09218 09217 09218 09217 09218 09221 09222 09223 09224 09222 09225 09226 09228 09228 09228 09229 09231 09231 09231 09232 09231	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA LHLD MOV ANI STA	Common\$Physical A,Floppy\$Read\$C I: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L Floppy\$Head A,L Floppy\$Track	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;;Get read function code ;Set command table ;Re-entry point to retry after error Type ;;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1 ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered ;Note: this is single byte value ; /2 for track (carry off from ANI above)</pre>
0BE7 0BEA 0BEC 0BF2 0BF2 0BF7 0BF9 0BF0 00BF0 00BF0 00C02 00C02 00C02 00C05 00C08 00C08 00C06 00C06 00C06 00C07 00000000	C3EC0B 3E01 321D0B 3A2A0A FE01 SE01 32320A C9 3A270A E601 321E0B 2A280A 7D E601 321F0B 7D 1F 32200B 3E01	09206 09207 09208 09209 09211 09212 09213 09214 09215 09215 09216 09217 09220 09221 09220 09220 09221 09222 09224 09225 09224 09227 09228 09227 09228 09227 09228 09227 09228 09227 09228 09227 09228 09229 09230	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA LHLD MOV ANI STA MOV RAR STA MVI	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L Floppy\$Track A,1	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; convert disk number to 0 or 1 ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered on head 1 ;Set head number ;Note: this is single byte value ; /2 for track (carry off from ANI above) ;Start with sector 1 as a whole</pre>
0BE7 0BEA 0BEC 0BF2 0BF2 0BF7 0BF9 0BF0 00BF0 00BF0 00C02 00C02 00C02 00C05 00C08 00C08 00C06 00C06 00C06 00C07 00000000	C3EC0B 3E01 321D0B 3A2A0A FE01 32320A C9 3A270A 5601 321E0B 2A280A 7D E601 321F0B 7D 1F 32200B	09206 09207 09208 09209 09210 09211 09212 09213 09214 09215 09216 09217 09218 09217 09218 09217 09218 09217 09218 09221 09223 09224 09223 09224 09225 09226 09227 09228 09223 09231 09233 09234 09235 09235 09235	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA LHLD MOV ANI STA	Common\$Physical A,Floppy\$Read\$C I: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L Floppy\$Head A,L Floppy\$Track	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;;Get read function code ;Set command table ;Re-entry point to retry after error Type ;;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1 ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered ;Note: this is single byte value ; /2 for track (carry off from ANI above)</pre>
0BE7 0BEA 0BEC 0BF2 0BF2 0BF7 0BF7 0BF7 0BF7 0BF7 0BF7 0BF7 0C02 0C02 0C02 0C02 0C05 0C08 0C08 0C06 0C06 0C06 0C06 0C06 0C06	C3EC0B 3E01 321D0B 3A2A0A FE01 3200 3200 32120B 2A280A 7D 1F 321F0B 7D 1F 322200B 3210B 321F0B 321F0B 321F0B 321000 321000 32100 3210000 3210000 321000	09206 09207 09208 09209 09211 09212 09213 09214 09215 09216 09217 09218 09217 09218 09217 09218 09219 09221 09221 09221 09221 09223 09224 09225 09225 09225 09226 09223 09228 09223 09231 09232 09234 09232 09234 09235 09236 09237	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA LHLD MOV ANI STA MOV RAR STA	Common\$Physical A,Floppy\$Read\$C I: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$S Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L Floppy\$Head A,L Floppy\$Track A,1 Floppy\$Sector	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;;Get read function code ;Set command table ;Re-entry point to retry after error Type ;;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1 ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered on head 1 ;Set head number ;Note: this is single byte value ;/2 for track (carry off from ANI above) ;Start with sector 1 as a whole ; track will be transferred ;</pre>
08E7 08EA 08EC 08F2 08F2 08F4 08F7 08F7 08F7 08F7 08F7 08F7 08F7 08F7	C3EC0B 3E01 321D0B 3A2A0A FE01 SE01 32320A C9 3A270A E601 321E0B 2A280A 7D E601 321F0B 7D 1F 32200B 3E01	09206 09207 09208 09209 09210 09211 09212 09213 09214 09215 09216 09217 09218 09217 09218 09217 09218 09217 09218 09221 09223 09224 09223 09224 09225 09226 09227 09228 09223 09231 09233 09234 09235 09235 09235	JMP Read\$Physical: MVI ; Common\$Physica STA ; Deblock\$Retry: LDA CPI JZ MVI STA RET Correct\$Disk\$T LDA ANI STA LHLD MOV ANI STA MOV RAR STA MVI	Common\$Physical A,Floppy\$Read\$C 1: Floppy\$Command In\$Buffer\$Disk\$ Floppy\$5 Correct\$Disk\$Ty A,1 Disk\$Error\$Flag ype: In\$Buffer\$Disk 1 Floppy\$Unit In\$Buffer\$Track A,L Floppy\$Track A,1	<pre>;Go to common code ;Read previously selected sector ; into disk buffer ode ;Get read function code ;Set command table ;Re-entry point to retry after error Type ;Get disk type currently in buffer ;Confirm it is a 5 1/4" floppy pe ;Yes ;No, indicate disk error ;Set up disk control table ; ;Convert disk number to 0 or 1 ; for disk controller ;Set up head and track number ;Even numbered tracks will be on ; head 0, odd numbered on head 1 ;Set head number ;Note: this is single byte value ; /2 for track (carry off from ANI above) ;Start with sector 1 as a whole ; track will be transferred ; ;Set byte count for complete</pre>

0C1E 0C21	21A40F 22240B	09243 09244	LXI SHLD	H,Disk\$Buffer Floppy\$DMA\$Addre	;Set transfer address to be ss ; disk buffer
		09245			;
		09246			;As only one control table is in
		09247			; use, close the status and busy ; chain pointers back to the
		09248 09249			; main control bytes
0024	214300	09250	LXI	H.Disk\$Status\$B]	
	22260B	09251	SHLD	Floppy\$Next\$Stat	
	214500	09252	LXI	H,Disk\$Control\$5	j _
0C2D	22280B	09253	SHLD	Floppy\$Next\$Cont	rol\$Location
		09254		· · · · · · · · · · · · · · · · · · ·	
	211D0B	09255	LXI	H,Floppy\$Command	;Set up command block pointer
0033	224600	09256	SHLD	Command\$Block\$5	
0036	214500	09257 09258	LXI	H.Disk\$Control\$5	Activate 5 1/4" disk controller
0039		09259	MVI	M. 80H	
		09260	;		
		09261	Wait\$For\$Disk\$C	omplete:	;Wait until disk status block indicates
		09262			; operation has completed, then check
		09263			; if any errors occurred.
		09264			;On entry HL -> disk control byte
OC3B		09265	XRA STA	A Disk\$Hung\$Flag	;Ensure hung flag clear
0030	32330A	09266 09267	aim	Prevention and a second	
0C3F	21570C	09268	LXI	H,Disk\$Timed\$Out	;Set up watchdog timer
	015802	09269	LXI	B,Disk\$Timer	;Time delay
	CD6D08	09270	CALL	Set\$Watchdog	
		09271	Disk\$Wait\$Loop:		
0048		09272	MOV	A, M	;Get control byte
0049		09273	ORA	A	de sus transformations
OC4A	CA5DOC	09274 09275	JZ	Disk\$Complete	;Operation done
OC4D	3A330A	09276	LDA	Disk\$Hung\$Flag	;Also check if time expired
0050		09277	ORA	A	Miso check in time capited
	C2B40D	09278	JNZ	Disk\$Error	;Will be set to 40H
		09279			
0054	C3480C	09280	JMP	Disk\$Wait\$Loop	
		09281	D/ -1 -T		- Control and the form webshire
		09282	Disk\$Timed\$Out:		;Control arrives here from watchdog ; routine itself so this is effectively
		09283 09284			; part of the interrupt service routine.
0057	3E40	09285	MVI	A.40H	;Set disk hung error code
0059	32330A	09286	STA	Disk\$Hung\$Flag	; into error flag to pull
		09287			; control out of loop
0050	C9	09288	RET		;Return to watchdog routine
		09289	D/-1-0		
ACED	010000	09290 09291	Disk\$Complete: LXI	B,0	;Reset watchdog timer
00.30	010000	09292		5,0	;HL is irrelevant here
0060	CD6D08	09293	CALL	Set\$Watchdog	
		09294			
	3A4300	09295	LDA	Disk\$Status\$Bloo	
	FE80	09296	CPI	80H	;Check if any errors occurred
0068	DAB40D	09297	JC .	Disk\$Error	;Yes
		09298 09299	; Disk\$Error\$Igno	re.	
0C6B	AF	09299	XRA	A	; No
0060	32320A	09301	STA	Disk\$Error\$Flag	;Clear error flag
0C6F	C9	09302	RET		-
		09303			
		09304	;		
		09400	;#		
		09401		ror message hand	ling
		09402 09403	1		
		09403	, Disk\$Error\$Mess	ages:	;This table is scanned, comparing the
		09405			; disk error status with those in the
		09406			; table. Given a match, or even when
		09407			; then end of the table is reached, the
		09408			; address following the status value
	••	09409		4011	; points to the correct message text.
0070		09410	DB	40H Dá slitting st 40	
	9D0C	09411	DW	Disk\$Msg\$40	
	41	09412	ne	41H	
0073	41 A200	09412 09413	DB DW	41H Disk\$Msg\$41	

Figure 8-10. (Continued)

0C76 42	09414	DB	42H	
0C77 AC0C	09415	DW	Disk\$Ms	- # 4 O
				J#72
0C79 21	09416	DB	21H	
OC7A BCOC	09417	DW	Disk\$Ms	3\$21
0C7C 22	09418	DB	22H	
OC7D C1OC	09419	DW	Disk\$Ms	#22
				J#22
0C7F 23	09420	DB	23H	
0080 0800	09421	DW	Disk\$Ms@	1523
0082 24	09422	DB	24H	
0C83 DA0C	09423	DW	Disk\$Ms(1\$24
0085 25	09424	DB	25H	
0C86 E60C	09425	DW	Disk\$Ms	172J
0088 11	09426	DB	11H	
0C89 F90C	09427	DW	Disk\$Ms	3\$11
OC8B 12	09428	DB	12H	
OC8C 070D	09429	DW	Disk\$Ms	3 \$12
0C8E 13	09430	DB	13H	
OC8F 140D	09431	DW	Disk\$Ms	as13
		DB		
0091 14	09432		14H	
0C92 220D	09433	DW	Disk\$Ms	3\$14
OC94 15	09434	DB	15H	
	09435	DW	Disk\$Ms	
0C95 310D				0141
OC97 16	09436	DB	16H	
0C98 3D0D	09437	DW	Disk\$Ms	\$16
		DB	0	
0C9A 00	09438			;<== Terminator
OC9B 4DOD	09439	DW	Disk\$Ms	s\$Unknown ;Unmatched code
	09440			
0000 -	09441	DEMAESA	EQU	3 :Disk error message table entry size
0003 =		DEM\$Entry\$Size	EQU	3 ;Disk error message table entry size
	09442	,		
	09443	: Message	texts	
	09444			
		7	-	
	E670009445	Disk\$Msg\$40:	DB	'Hung',0 ;Timeout message
0CA2 4E6F7	4205209446	Disk\$Msg\$41:	DB	'Not Ready',0
0CAC 57726	9746509447	Disk\$Msg\$42:	DB	'Write Protected',0
			DB	'Data',0
	4610009448	Disk\$Msg\$21:		
0CC1 466F7	26D6109449	Disk\$Msg\$22:	DB	'Format',O
0008 40697	3736909450	Disk\$Msg\$23:	DB	'Missing Data Mark',0
OCDA 42757		Disk\$Msg\$24:	DB	'Bus Timeout',0
0CE6 436F6	E747209452	Disk\$Msg\$25:	DB	'Controller Timeout',0
OCE9 44726	9766509453	Disk\$Msg\$11:	DB	'Drive Address',0
			DB	'Head Address',0
OD07 48656		Disk\$Msg\$12:		
0D14 54726	1636B09455	Disk\$Msg\$13:	DB	'Track Address',0
0D22 53656	3746E09456	Disk\$Msg\$14:	DB	'Sector Address',0
0D31 42757		Disk\$Msg\$15:	DB	'Bus Address',0
0D3D 496C6	C656709458	Disk\$Msg\$16:	DB	'Illegal Command',0
0D4D 556E6	B6E6F09459	Disk\$Msg\$Unknows	1:	DB 'Unknown',0
	09460	•		
		<u>*</u>		
	09461	Disk\$EM\$1:		;Main disk error message part 1
0055 07000	A 09462		DB	BELL, CR, LF
OD58 44697			DB	'Disk ',0
0000 4407/		-		weeks yw
	09464	;		· · · · · · ·
	09465			;Error text output next
	09466			
	09467	, Disk\$EM\$2:		;Main disk error message part 2
		DISK PENPZI		
OD5E 20457			DB	1 Error (1
OD66 0000	09469	Disk\$EM\$Status:	DB	0,0 ;Status code in Hex.
0D68 290D0			DB	<pre>/)',CR,LF,' Drive '</pre>
0000 27000	00474	Distant	DB	
OD76 00	09471	Disk\$EM\$Drive:		
0D77 2C204	8656109472		DB	<pre>/, Head /</pre>
0D7E 00	09473	Disk\$EM\$Head:	DB	0 ;Head number
0D7F 2C205			DB	', Track '
0D87 0000	09475	Disk\$EM\$Track:	DB	0,0 ;Track number
0D89 2C205	3656309476		DB	1, Sector 1
0092 0000	09477	Disk\$EM\$Sector:		0,0 ;Sector number
		DISKACHASector:	00	
0D94 2C204	F706509478		DB	<pre>/, Operation - /</pre>
0DA2 00	09479		DB	0 ;Terminator
	09480			
				(D. 1. (. D
0DA3 52656	1642E09481	Disk\$EM\$Read:	DB	'Read.',0 ;Operation names
	9746509482	Disk\$EM\$Write:	DB	'Write.',0
	09483	;		··· · ·
	09484	7		
	09485	Disk\$Action\$Con	firm:	
			DB	0 ;Set to character entered by user
	09494			
ODBO OO	09486		DD	
ODBO OO ODB1 ODOAO	0 09487		DB	CR,LF,O
		;	DB	CR, LF, O
	0 09487 09488			
	0 09487		DB ror proce	

		·		·····					
	09490	;							
	09491		nis rout	ine builds and outputs	s an error message.				
	09492	; Th	The user is then given the opportunity to:						
	09493	;							
	09494	;	R	retry the operatio	on that caused the error				
	09495	;			and attempt to continue				
	09496	;	A	abort the program	and return to CP/M.				
	09497	;							
	09498	Disk\$Error							
ODB4 F5	09499				rve error code from controller				
ODB5 21660D	09500	LX		,Disk\$EM\$Status	;Convert code for message				
ODBS CD440E	09501	CA	ALL C	AH	;Converts A to hex.				
	09502								
ODBB 3A270A	09503	LD		n\$Buffer\$Disk	;Convert disk id. for message				
ODBE C641	09504	AD		A*	;Make into letter				
ODCO 32760D	09505	ST	D A1	isk\$EM\$Drive					
	09506								
ODC3 3A1FOB	09507	LD		loppy\$Head	;Convert head number				
ODC6 C630	09508	AD		01					
ODC8 327EOD	09509	ST	FA D	isk\$EM\$Head					
	09510								
ODCB 3A200B	09511	LD		loppy\$Track	;Convert track number				
ODCE 21870D	09512	LX		,Disk\$EM\$Track					
ODD1 CD440E	09513	CA	ALL C	AH					
	09514								
ODD4 3A210B	09515	LD		loppy\$Sector	;Convert sector number				
ODD7 21920D	09516	LX		,Disk\$EM\$Sector					
ODDA CD440E	09517	CA	ALL C	AH					
	09518								
ODDD 21550D	09519	LX		,Disk\$EM\$1	;Output first part of message				
ODE0 CD5305	09520	CA	ALL O	utput\$Error\$Message					
	09521								
ODE3 F1	09522	PO		SW	Recover error status code;				
ODE4 47	09523	MC)V B	, A	;For comparisons				
ODE5 216DOC	09524	LX	кі н	,Disk\$Error\$Messages	- DEM\$Entry\$Size				
	09525				;HL -> table - one entry				
ODE8 110300	09526	LX		,DEM\$Entry\$Size	;Get entry size for loop below				
	09527	Disk\$Error							
ODEB 19	09528	DA	AD D		;Move to next (or first) entry				
	09529								
ODEC 7E	09530	MO		, M	;Get code number from table				
ODED B7	09531	OR			Check if end of table				
ODEE CAF80D	09532	JZ		isk\$Error\$Matched	;Yes, pretend a match occurred				
ODF1 B8	09533	CM			;Compare to actual code				
ODF2 CAF80D	09534	JZ		isk\$Error\$Matched	;Yes, exit from loop				
ODF5 C3EBOD	09535	JM	ים אוי	isk\$Error\$Next\$Code	;Check next code				
	09536								
	09537	Disk\$Error			-10 X				
0DF8 23	09538	IN			;HL -> address of text				
ODF9 5E	09539			, M	;Get address into DE				
ODFA 23	09540	IN							
ODFB 56	09541	MO		, M	-18				
ODFC EB	09542		CHG		;HL -> text				
0DFD CD5305	09543	CA	ALL O	utput\$Error\$Message	;Display explanatory text				
	09544			Distant CM40	Display assess such of march				
0E00 215E0D	09545	LX		,Disk\$EM\$2	;Display second part of message				
0E03 CD5305	09546	CA	ALL O	utput\$Error\$Message					
0504 014005	09547				Choose operation toxt				
0E06 21A30D	09548	LX	чт н	,Disk\$EM\$Read	;Choose operation text ; (assume a read)				
0500 341000	09549	LD		loppy\$Command	; (assume a read) ;Get controller command				
OEO9 3A1DOB	09550 09551	CP		loppy\$Command loppy\$Read\$Code	yoet controller command				
OEOC FEO1				loppy\$Kead\$Code isk\$Error\$Read	;Yes				
0E0E CA140E 0E11 21A90D	09552 09553	JZ		isk\$Error\$Kead ,Disk\$EM\$Write	;res ;No, change address in HL				
UCII ZIMTUD	09553	Disk\$Error		, DISKACLIAMLICE	had change address in ht				
0E14 CD5305	09555			utput\$Error\$Message	;Display operation type				
V214 CD0300	09555		0	arearerior enessage	Propras Operation type				
	09556	; Disk\$Error	t Paris -	+ SAction:	;Ask the user what to do next				
0517 000505					;Ask the user what to do next ;Display prompt and wait for input				
0E17 CD2F05	09558	ĽA	ALL R	equest\$User\$Choice	; Returns with A = uppercase char.				
	09559	CF	- TC	R ⁷	; Returns with A = uppercase char. ;Retry?				
OE1A FE52	09560 09561	UF 12		ĸ∵ isk\$Error\$Retry	, , , , , , , , , , , , , , , , , , ,				
OE1C CA2COE	07361			iskæerroræketry A'	; Abort				
0515 5541	00540								
0E1F FE41	09562	CF			, ADOT (
0E21 CA360E	09562 09563	JZ	z s	ystem\$Reset					
	09562		z s		;Ignore				

Figure 8-10. (Continued)

0E29 C3170E	09566 09567	•	JMP	Disk\$Error\$Requ	est\$Action
	09568	, Diebser	ror\$Retr	v.	;The decision on where to return
	09569	DISKALI	i or ene ci	<i>,.</i>	; depends on whether the operation
	09570				; failed on a deblocked or
	09571				<pre>; nondeblocked drive.</pre>
0E2C 3A350A	09572		LDA	Selected\$Disk\$D	eblock
OE2F B7	09573		ORA	Α	
0E30 C2EF0B	09574		JNZ	Deblock\$Retry	
0E33 C33C0B	09575		JMP	No\$Deblock\$Retr	v
	09576				
	09577	, System\$1	Pecat.		;This is a radical approach, but
	09578	Systemp	Reseti		; it does cause CP/M to restart.
0E36 0E00	09579		MVI	C, O	;System reset
0E38 CD0500	09580		CALL	BDOS	
	09581				
	09582	;			
	09583	;			
	09584	;	A to up	Der	
	09585	, ,	II to ap		
	09586	;	Convert	s the contents o	f the A register to an upper-
	09587	;			rrently a lowercase letter.
		•	case le	iter in it is cu	rientij a lowercase letter.
	09588	;			
	09589	;	Entry P	arameters	
	09590	;			
	09591	;		A = character t	o be converted
	09592	;			
	09593	;	Exit pa	rameters	
	09594	;			
	09595			A	haracter
		;		A = converted c	narauter
	09596	;			
	09597	A\$To\$Up			
OE3B FE61	09598		CPI	'a'	;Compare to lower limit
OE3D D8	09599		RC		;No need to convert
OE3E FE7B	09600		CPI	'z' + 1	;Compare to upper limit
0E40 D0	09601		RNC		;No need to convert
0E41 E65F	09602		ANI	5FH	;Convert to uppercase
0E43 C9	09603		RET	0.11	yountere to appendate
0E43 C7		_			
	09604	7	~ ·		
	09605	7	Convert	A register to h	exadecimal
	09606	;			
	09607	;	This su	broutine convert	s the A register to hexadecimal.
	09608	;			
	09609	;	Entry p	arameters	
	09610				
	09611	;		A = value to be	converted and output
	09612				ea to receive two characters of output
		7		nc -/ purrer ar	ea to receive two characters of output
	09613	;			
	09614	;	Exit pa	rameters	
	09615	;			
	09616	;		HL -> byte foll	owing last hex byte output
	09617	;			
	09618	CAH:			
0E44 F5	09619		PUSH	PSW	;Take a copy of the value to be converted
0E45 0F	09620		RRC		;Shift A right four places
			RRC		, II I ADIL I CAI FAGES
OE46 OF	09621				
OE47 OF	09622		RRC		
0E48 OF	09623		RRC		
OE49 CD4DOE	09624		CALL	CAH\$Convert	;Convert to ASCII
OE4C F1	09625		POP	PSW	;Get original value again
	09626				;Drop into subroutine, which converts
	09627				; and returns to caller
	09628	CAH\$Con	vert.		,
0E4D E60F	09629	0004000	ANI	0000\$1111B	;Isolate LS four bits
0E4F C630	09629		ADI	'0'	Convert to ASCII
OE51 FE3A	09631		CPI	191 + 1	;Compare to maximum
0E53 DA580E	09632		JC	CAH\$Numeric	;No need to convert to A -> F
0E56 C607	09633		ADI	7	;Convert to a letter
	09634	CAH\$Num			
0E58 77	09635		MOV	M, A	;Save character
0E59 23	09636		INX	н	;Update character pointer
	09637		RET		telere evenance, between
0554 09			AL I		
0E5A C9					
0E5A C9	09638				
0E5A C9	09639	;			
0E5A C9		; ; ;#			

	09701 09702		ontrol table image	s for warm boot
	09703 09704	; Dest#Conturit®:		
0E5B 01	09704	Boot\$Contro1\$Pa DB	1	;Read function
0E5C 00	09706	DB	0	;Unit (drive) number
0E5D 00	09707	DB	ŏ	;Head number
0E5E 00	09708	DB	ŏ	Track number
0E5F 02	09709	DB	2	;Starting sector number
0E60 0010	09710	กัพ	- 8×512	;Number of bytes to read
0E62 00C4	09711	DW	CCP\$Entry	;Read into this address
0E64 4300	09712	DW	Disk\$Status\$Bloc	
0E66 4500	09713	DW	Disk\$Control\$5	Pointer to next control table
	09714	Boot\$Control\$Pa		· · · · · · · · · · · · · · · · · · ·
0E68 01	09715	DB	1	;Read function
0E69 00	09716	DB	0	;Unit (drive) number
0E6A 01	09717	DB	1	;Head number
0E6B 00	09718	DB	0	;Track number
0E6C 01	09719	DB	1	;Starting sector number
0E6D 0006	09720	DW	3*512	Number of bytes to read
0E6F 00D4	09721	DW	CCP\$Entry + (8×5	(2) Read into this address
DE71 4300	09722	DW	Disk\$Status\$Bloc	
DE73 4500	09723	DW	Disk\$Contro1\$5	;Pointer to next control table
	09724			
	09725	;		
	09726	;		
	09800	; #		
	09801	;		
	09802	WBOOT:	;Warm boot entry	
	09803		;On warm	boot, the CCP and BDOS must be reloaded
	09804			nemory. In this BIOS, only the 5 1/4"
	09805		; diske	tes will be used, therefore this code
	09806			dware specific to the controller. Two
	09807			pricated control tables are used.
DE75 318000	09808	LXI	SP,80H	
0E78 115B0E	09809	LXI	D,Boot\$Control\$P	
DE7B CD8AOE	09810	CALL	Warm\$Boot\$Read	;Load drive 0, track 0,
	09811			; head 0, sectors 2 - 8
DE7E 11680E	09812	LXI	D,Boot\$Contro1\$P	
0E81 CD8AOE	09813	CALL	Warm\$Boot\$Read	;Load drive 0, track 0,
	09814			; head 1, sectors 1 - 3
DE84 CDDFOE	09815	CALL	Patch\$CPM	;Make custom enhancements patches
DE87 C36C02	09816	JMP	Enter\$CPM	;Set up base page and enter CCP
	09817	; 		On entry, DE -> control table image
	09818 09819	Warm\$Boot\$Read:		This control table is moved into
	09819			the main disk control table and
	09820			then the controller activated.
		1 * *		;HL -> actual control table
DESA 211DOB	09822		H,Floppy\$Command Command\$Block\$5	
DE8D 224600	09823	SHLD	Command#Block#J	;Tell the controller its address
	09824			;Move the control table image
	09825 09826	MVI	C.13	; into the control table itself. Set byte count
DE90 OEOD				set byte count
NC02 14	09827	Warm\$Boot\$Move:		Get image byte
DE92 1A	09828 09829	LDAX MOV		Store into actual control table
DE93 77 DE94 23	09829	INX		Update pointers
DE94 23 DE95 13	09830	INX	D	opuale pointers
0E96 0D	09832	DCR		Count down on byte count
0E97 C2920E	09832	JNZ		Continue until all bytes moved
UE7/ UZ72UE	09833	UNZ	Har mapoot anove	CONTINUE OUTIL GIT DYLES MOVED
0E9A 214500	09835	LXI	H,Disk\$Control\$5	;Activate controller
0E9D 3680	09836	MVI	M.80H	,
	09837	Wait\$For\$Boot\$C		
0E9E 7E	09838	MOV	A.M	:Get status byte
DEAD B7	09839	ORA	A	;Check if complete
OEA1 C29FOE	09840	JNZ	Wait\$For\$Boot\$Co	
	09841			;Yes, check for errors
0EA4 3A4300	09842	LDA	Disk\$Status\$Bloc	
0EA7 FE80	09843	CPI	80H	•
DEA9 DAADOE	09844	JC	Warm\$Boot\$Error	;Yes, an error occurred
OEAC C9	09845	RET		,,
	09846	1		
	09847	, Warm\$Boot\$Error	· ;	
OEAD 21B60E	09848	LXI	H.Warm\$Boot\$Erro	*SMessage
OEBO CD5F02	09849	CALL	Display\$Message	

Figure 8-10. (Continued)

0EB3 C3750E	09850 09851	JMP	WBOOT		;Restart warm boot
	09852	; Warm\$Boot\$Erro	r\$Message		
OEB6 0D0A57617		DB			t Error - retrying′,CR,LF,O
	09854	;			
	09855	;			
	10000	÷#			
	10001				
	10002	Ghost\$Interrup	t:	:Contro	I will only arrive here under the most
	10003				ual circumstances, as the interrupt
	10004				roller will have been programmed to
	10005				ress unused interrupts.
	10006				
OED8 F5	10007	PUSH	PSW		;Save pre-interrupt registers
0ED9 3E20	10008	MVI	A, IC\$E	זכ	;Indicate end of interrupt
OEDB D3D8	10009	OUT	IC\$OCW2		,
OEDD F1	10010	POP	PSW		
OEDE C9	10011	RET			
	10012	;			
	10013	;			
	10100	;*			
	10101	;			
	10102	; Patch	CP/M		
	10103	7			
	10104		outine ma	akes some	very special patches to the
	10105				o make some custom enhancements
	10106				
	10107	; Public	files:		
	10108	;		ge hard d	isk systems it is extremely useful
	10109	7	to part	tition th	e disk using the user number features.
	10110		However	, it bec	omes wasteful of disk space because
	10111	;			of common programs must be stored in
	10112				This patch makes User O public
	10113	7	accessi	ible from	any other user area.
	10114	;		RNING ***	
	10115	7	Files i	in User (MUST be set to system and read/only
	10116	;	status	to avoid	their being accidentally damaged.
	10117	;	Because	e of the	side effects associated with public
	10118	;			h can be turned on or off using
	10119	;	a flag	in the 1	ong term configuration block.
	10120	;			
	10121	; User p			
	10122	;			1's USER command and user numbers
	10123	;			is all too easy to become confused
	10124	;	and for	rget whic	h user number you are "in." This
	10125	;	patch r	nodifies	the CCP to display a prompt which
	10126	;			the default disk id., but also the
	10127	Ŧ			mber, and an indication of whether
	10128	;	public	files an	e enabled:
	10129	;			
	10130	;			P3B> or 3B>
	10131	;			A
	10132	;			When public files are enabled.
	10133	;			
	10134	; Equate	s for put	olic file	5
	10135	;			
D35E =	10136	PF\$BDOS\$Exit\$P		EQU	BDOS\$Entry + 758H
D37C =	10137	PF\$BDOS\$Char\$M		EQU	BDOS\$Entry + 776H
D361 =	10138	PF\$BDOS\$Resume		EQU	BDOS\$Entry + 75BH
000D =	10139	PF\$BDOS\$Unused	\$Bytes	EQU	13
	10140	;			
	10141	;	_		
	10142	; Equate	s for use	er prompt	
	10143	;			
C788 =	10144	UP\$CCP\$Exit\$Po		EQU	CCP\$Entry + 388H
C78B =	10145	UP\$CCP\$Resume\$		EQU	CCP\$Entry + 38BH
C513 =	10146	UP\$CCP\$Get\$Use		EQU	CCP\$Entry + 113H
C5D0 =	10147	UP\$CCP\$Get\$Dis	k\$Id	EQU	CCP\$Entry + 1DOH
C48C =	10148	UP\$CCP\$CONOUT		EQU	CCP\$Entry + 8CH
	10149	;			
	10150	;			
	10151		the inte	ervention	points
	10152	;			
	10153	Patch\$CPM:			
0EDF 3EC3 0EE1 325ED3	10154	MVI	A, JMP		;Set up opcode

0EE4 3288C	7 10156	STA	UP\$CCP\$Exit\$Poin	nt		
0EE7 21F40		LXI	H,Public\$Patch			
OEEA 225FD		SHLD	PF\$BDOS\$Exit\$Point + 1			
OEED 21110	F 10159	LXI	H,Prompt\$Patch ;Get address of intervening code			
0EF0 2289C		SHLD	UP\$CCP\$Exit\$Poin	nt + 1		
	10161					
0EF3 C9	10162	RET		;Return to enter CP/M		
	10163	;				
	10164	;				
	10165	;				
	10166	Public\$Patch:		;Control arrives here from the BDOS		
	10167			;The BDOS is in the process of scanning		
	10168			; down the target file name in the : search next function		
	10169					
	10170 10171					
	10171			; DE -> directory entry : B = character count		
	10172			, D = character count		
0EF4 3A420		LDA	CR&Public&Files	;Check if public files are to be enabled		
0EF7 B7	10175	ORA	A	Check in public files are to be enabled		
OEF8 CAOBO		JZ	No\$Public\$Files	:No		
VERO CAVBU	10176	52	HOFF GUILCHIILES			
OEFB 78	10178	MOV	A,B	;Get character count		
OEFC B7	10179	ORA	A, 5	;Check if looking at first byte		
	10180	UNH	••	; (that contains the user number)		
OEFD C20B0		JNZ	No\$Public\$Files	:No, ignore this patch		
	10182					
0F00 1A	10183	LDAX	D	;Get user number from directory entry		
OF01 FEE5	10184	CPI	0E5H	;Check if active directory entry		
OFO3 CAOBO	F 10185	JZ	No\$Public\$Files	;Yes, ignore this patch		
	10186					
0F06 7E	10187	MOV	Α,Μ	;Get user number		
OF07 B7	10188	ORA	Α	;Check if User O		
OFO8 CA7CD		JZ	PF\$BDOS\$Char\$Ma	tches ;Force character match		
	10190			_		
	10191	No\$Public\$Files		Replaced patched out code		
OFOB 78	10192	MOV	A,B	;Check if count indicates that		
OFOC FEOD	10193	CPI	PF\$BD0S\$Unused\$1			
	10194			; unused bytes field of FCB		
OFOE C361D	3 10195 10196	JMP	PF\$BDOS\$Resume\$	°oint ;Return to BDOS		
	10196 10197	; Prompt\$Patch:		;Control arrives here from the CCP		
	10197	rrompt#ratch:		;Control arrives nere from the CCP ;The CCP is just about to get the		
	10198			; drive id. when control gets here.		
	10200			The CCP's version of CONOUT is used		
	10200			; so that the CCP can keep track of		
	10202			; the cursor position.		
	10203					
0F11 3A420		LDA	CB\$Public\$Files	;Check if public files are enabled		
OF14 B7	10205	ORA	A			
OF15 CA1DO		JZ	UP\$Private\$File	s ;No		
	10207					
0F18 3E50	10208	MVI	A, 'P'			
OF1A CD8CC		CALL	UP\$CCP\$CONOUT	;Use CCP's CONOUT routine		
	10210					
	10211	UP\$Private\$File				
OF1D CD13C		CALL		;Get current user number		
OF20 FEOA	10213	CPI	9 + 1	;Check if one or two digits		
0F22 D2300		JNC	UP\$2\$Digits /0/	0		
0F25 C630	10215	ADI	.0.	;Convert to ASCII		
	10216	UP\$1\$Digit:	URACORECONOUT	· Dutout the abayantay		
OF27 CD8CC OF2A CDD0C		CALL	UP\$CCP\$CONOUT UP\$CCP\$Get\$Disk	;Output the character \$Id ;Get disk identifier		
OF 2A CDDOC		JMP	UP\$CCP\$Resume\$P			
0720 03880	10219	e cont	or your ane sume ar	orne yneeden eo oor		
	10220	, UP\$2\$Digits:				
0F30 C626	10222	ADI	101 - 10	Subtract 10 and convert to ASCII		
0F32 F5	10223	PUSH	PSW	;Save converted second digit		
0F33 3E31	10224	MVI	A, 11	;Output leading '1'		
0F35 CD8CC		CALL	UP\$CCP\$CONOUT			
0F38 F1	10226	POP	PSW	Recover second digit		
0F39 C3270		JMP	UP\$1\$Digit	;Output remainder of prompt and return to		
	10228		- • •	; the CCP		
1	10229					
	10230	;				
	10300	;#				

Figure 8-10. (Continued)

			· · · · · · · · · · · · · · · · · · ·	
	10301			
1		;		
	10302	; Conf:	iguration block get addre	255
	10303	;		
1	10304	; This	routine is called by uti	lity programs running in the TPA.
	10305	: Give	a specific code number.	it returns the address of a specific
1	10306		t in the configuration b	
1			it in the configuration b	JICCK.
	10307	;		
	10308	; Byu	sing this routine, utilit	y programs need not know the exact
	10309	; layou	ut of the configuration b	lock.
	10310			
	10311		/ parameters	
			Palameters	
	10312	;		
	10313	;	C = Object identity c	ode (in effect, this is the
1	10314	;	subscript of	the object's address in the
	10315	;	table below)	
	10316			
	10317	,	*******	
1				· · · · · · · · · · · · · · · · · · ·
	10318	CB\$Get\$Addre	51	;<=== BIOS entry point (private)
	10319			
OF3C F5	10320	PUSH	PSW	;Save user's registers
OF3D C5	10321	PUSH	В	
OF3E D5	10322	PUSH	D	
1	10323	1 0011	-	
0505 (0		MOU		Material fato a contrat
0F3F 69	10324	MOV	L,C	;Make code into a word
0F40 2600	10325	MVI	н,о	
OF42 29	10326	DAD	н	<pre>;Convert code into word offset</pre>
0F43 114F0F	10327	LXI	D,CB\$Object\$Table	;Get base address of table
0F46 19	10328	DAD	D	;HL -> object's address in table
0F47 5E		MOV		
	10329		E,M	;Get LS byte
0F48 23	10330	INX	н	
OF49 56	10331	MOV	D, M	;Get MS byte
OF4A EB	10332	XCHG		;HL = address of object
	10333			,
OF4B D1	10334	POP	D	·Persuan unaria variatore
				;Recover user's registers
OF4C C1	10335	POP	В	
OF4D F1	10336	POP	PSW	
	10337			
OF4E C9	10338	RET		
	10339	:		
	10400			
		; #		
	10401	;		
1	10402	CB\$Object\$Tat	ole:	
	10403		:	Code
	10404		;	vv
OF4F 8FOF	10405	nw	Date	
				;01 date in ASCII
0F51 990F	10406	DW	Time\$In\$ASCII	;02 time in ASCII
0F53 A30F	10407	DW	Time\$Date\$Flags	;03 flags indicated if time/date set
OF55 8DOF	10408	DW	CB\$Forced\$Input	;04 forced input pointer
0F57 4300	10409	DW	CB\$Startup	;05 system startup message
1	10410		Seferal rap	; Redirection words
0F59 5800		υM	CD#Cl-#I	
	10411	D	CB\$Console\$Input	;06
0F5B 5A00	10412	DW	CB\$Console\$Output	;07
0F5D 5C00	10413	DW	CB\$Auxiliary\$Input	;08
0F5F 5E00	10414	DW	CB\$Auxiliary\$Output	:09
0F61 6000	10415	DW	CB\$List\$Input	;10
0F63 6200	10416	DW	CB\$List\$Output	;11
0100 0200	10417	DW	CD7LIS(#Od(Pd)	y 4 4
0F65 6400	10418	DW	CB\$Device\$Table\$Addre	
0F67 B500	10419	DW	CB\$12\$24\$Clock	;13 Selects 12/24 hr. format clock
0F69 BD00	10420	DW	RTC\$Ticks\$per\$Second	;14
OF6B BF00	10421	DW	RTC\$Watchdog\$Count	;15
0F6D C100	10422	DW	RTC\$Watchdog\$Address	;16
0F6F C300	10423	DW	CB\$Function\$Key\$Table	
0F71 1B02	10424	DW	CONOUT\$Escape\$Table	;18
	10425			
0F73 8400	10426	DW	DO\$Initialize\$Stream	;19
0F75 9100	10427	DW	DO\$Baud\$Rate\$Constant	:20
0F77 9400	10428	DW	D1\$Initialize\$Stream	;21
0F79 A100	10429	DW	D1\$Baud\$Rate\$Constant	
0F7B A400	10430	DW	D2\$Initialize\$Stream	;23
0F7D B100	10431	DW	D2\$Baud\$Rate\$Constant	
0F7F 4002	10432	DW	Interrupt\$Vector	;25
0F81 890F	10433	DW	LTCB\$Offset	; 26
OF83 8BOF	10434	DW	LTCB\$Length	;27
0F85 4200		DW		
0685 4200	10435	DW	CB\$Public\$Files	; 30
1				

0F87								
	A421	10436		DW	Multi\$Co	ommand\$Bu	iffer ;31	
		10437	;				·	
		10500	, ; #					
		10501	;	The sho	rt term d	onfigura	ation block.	
		10502	;			-		
		10503	;	This con	ntains va	ariables	that can be set once CP/M	
		10504	;	has been	n initiat	ed, but	that are never preserved	
		10505	;	from one	e loading	of CP/M	1 to the next. This part of	
		10506		the con-	figuratio	on block	form the last initialized bytes	
		10507	;	in the l				
		10508	;					
		10509	;				e used by utility programs that	
		10510	;				; term configuration block from disk.	
		10511	;				5-byte page boundary, and therefore	
		10512	,	will als	ways be d	on a 128-	byte sector boundary in the reserved	
		10513	,	area on	the disk	<. A util	lity program can then, using the	
		10514	;	CB\$Get\$	Address F	Private E	BIOS call, determine how many 128-byte	
		10515	;	sectors	need to	be read	in by the formula:	
		10516	;					
		10517	,		(LCTB\$01	fset + L	TCB\$Length) / 128	
		10518	;					
		10519	;				offset from the start of the BIOS to	
		10520	;				the long term configuration block	
		10521	;				and the length, the utility can	
		10522	,				the LTCB over the disk image	
		10523	,	that it	has read	i from th	ne disk, and then write the	
		10524	;	updated	LTCB bad	sk onto t	the disk.	
		10525	;					
0F89		10526	LTCB\$Of 1		DW		ry - Long\$Term\$CB	
0F8B		10527	LTCB\$Ler	ngth:	DW	Long\$Ter	m\$CB\$End - Long\$Term\$CB	
		10528	,	F	4			
		10529	;	Forced	input poi	Inter		
		10530	1	14 000			this pointer is pointing to a source-	
		10531	;				this pointer is pointing to a nonzero	
		10532	7	oyte, ti	nen this	oyte wil	I be injected into the console input	
		10533	;	stream a	as though	1 IT had	been typed on the console. The to the next byte in memory.	
		10534	;	pointer	is (nen	upoated	to the next byte in memory.	
0000		10535	CREE			DU	CRECtartum	
0F8D		10536	CB\$Force	ru⇒input	•	DW	CB\$Startup	
		10537 10538	;					
		10538	; Date:			+ Current	system date	
AFOF	31302F3137		Dates	DB	10/17/8		;Unless otherwise set to the contrary	
VI OF		10540		~~	10/1//0	- ,	; this is the release date of the system	
		10542					;Normally, it will be set by the DATE utili	ty
0F98		10543		DB	0		;00-byte terminator	•
5, 70		10544			-		,,. .	
		10545	Time\$in\$	ASCIL		:Current	system time	
0F99		10546	HH:	DB	1001		;Hours	
OF9B		10547		DB	· · ·		•	
OF9C		10548	MM:	DB	1001		;Minutes	
OF9E		10549		DB	1			
OF9F		10550	SS:	DB	1001		; Seconds	
		10551		ASCIISE	nd:		;Used when updating the time	
OFA1	0A	10552		DB	LF			
0FA2		10553		DB	0		;00-byte terminator	
		10554	;					
		10555	;					
		10556	Time\$Dat	te\$Flags	:	;This by	te contains two flags that are used	
		10557					ndicate whether the time and/or date	
		10558				; have	been set either programmatically or	
		10559					ing the TIME and DATE utilities. These	
		10560					s can be tested by utility programs that	
		10561				; need	to have the correct time and date set.	
OFA3		10562		DB	0			
0001		10563	Time\$Se		EQU	0000\$000		
		10564	Date\$Set	1	EQU	0000\$001	10B	
0002		10565						
		10566	;					
			; #					
		10700	, .					
		10701	;	Uniniti	alized bu	Tilet ale	245	
		10701 10702	;					
		10701 10702 10703	; ; ;	With th	e except:	ion of th	ne main Disk\$Buffer, which contains a few	
		10701 10702 10703 1070 4	; ; ;	With th bytes o	e except: f code, a	ion of th all of th	ne main Disk\$Buffer, which contains a few ne other uninitialized variables	
		10701 10702 10703	; ; ;	With the bytes o occur h	e except: f code, a ere. This	ion of th all of th s has the	ne main Disk\$Buffer, which contains a few	

	10707	; since	uninitia	alized a	reas do I	not need t	o be kept on the disk.
	10708	;					
1	10709	;					
	10800	, ;#					
	10801	1					
	10802		1d boot	initial	ization	code is o	nly needed once.
	10803						executed.
1	10803		i be ove	rwritten	Unce it	Has been	ain disk buffer.
	10804	; Theref	ore, it	15 010	den ins	ICE CIE M	ain disk burrer.
		,					
0514	10806	7	DS	.			
OFA4	10807	Disk\$buffer:	bs	Physi	cal\$Sect	or\$Size *	Physical\$Sec\$Per\$Track
	10808	;				-	
	10809						the location counter
21A4 =	10810	After\$Disk\$Buf	fer	EQU	\$;\$ = ci	urrent value of location counter
	10811	;					
OFA4	10812			ORG	Disk\$	Buffer	;Wind the location counter back
	10813	;					
	10814	Initialize\$Str	'eam:				s used by the
	10815					subrouti	ne. It has the following
	10816			; fo	rmat:		
	10817			;			
	10818			;	DB		umber to be initialized
	10819			;	DB		of byte to be output
	10820			;	DB	××,××,	xx,xx data to be output
1	10821			;	:		
1	10822			;	:	_	
	10823			;	DB	Port n	umber of OOH terminates
	10824			;			
	10825			;			
	10826	;					
	10827				declare		
OFA4 D8	10828	DB		W1\$Port	;Prog	ram the 8	259 interrupt controller
OFA5 01	10829	DB	1				
OFA6 56	10830	DB	IC\$IC	W1			
	10831						
OFA7 D9	10832	DB DB		W2\$Port			
0FA8 01	10833 10834	DB	1 IC\$IC				
0FA9 02	10834	DB	10\$10	W2			
OFAA D9	10835	DB	TOROC	W1\$Port			
OFAB 01	10838	DB	1000	WIDFORU			
OFAC FC	10838	DB	IC\$OC				
OTHETE	10839	66	10400	~1			
OFAD 83	10840	DB	83H			Progr	am the 8253 clock generator
OFAE 01	10841	DB	1			,11091	am the 0200 clock generator
OFAF 34	10842	DB	-	\$010\$0B		:Count	er O, periodic interrupt, mode 2
	10843						
OFBO SO	10844	DB	80H			RTC u	ses channel O
OFB1 02	10845	DB	2				
OFB2 0146	10846	DW	17921			;19721	* 930 nanoseconds =
	10847					; 16.0	666 milliseconds). 60 ticks/sec.
0FB4 00	10848	DB	-0		;Port		f O terminates
	10849	;			-		
	10850	;					
	10851	Signon\$Message	:				
0FB5 43502F4D		DB		2.2.			
OFBE 3030	10853	DW	VERSI		;Curr	ent versi	on number
OFCO 20	10854	DB	11				
0FC1 3032	10855	DW	MONTH		;Curr	ent date	
OFC3 2F	10856	DB	111		· ·		
0FC4 3236	10857	DW	DAY				
OFC6 2F	10858	DB	111				
0FC7 3833	10859	DW	YEAR				
OFC9 ODOAOA	10860	DB	CR,LF				
0FCC 456E6861		DB			S′,CR,LF		
0FDC 4469736B		DB		Configu	ration :	1, CR, LF, LI	=
0FF3 20202020		DB				" Floppy'	
1011 20202020		DB	1			" Floppy'	
1030 20202020		DB	1	C: 0.24	Mbyte 8	" Floppy',	, CR, LF
104E 20202020	2010866	DB	1			" Floppy'	
106C 20202020		DB	'	M: 0.19	Mbyte M	emory Disl	k',CR,LF,LF
1	10868	;					
108D 00	10869	DB	0				
1	10870	;					
	10871		es for l	M\$Disk			
1	10872	7					

		10873	M\$Disk\$Se	tup\$Me			
108E	2020202020		5	B	M\$Dis	k already c	ontains valid information.',CR,LF,O
		10875			\$Message:		
1000	2020202020	010876 10877	, ")B	* M\$Dis		initialized to empty state.',CR,LF,O
		10878	M\$Disk\$Di	r\$Entr:	y:	;Dummy	directory entry used to determine
		10879				; if	the M\$Disk contains valid information
10F3		10880		B	15	`;User	15
	4D2444697			B	M\$Disk		
	A0A020	10882)B)B		F80H,	;System and read/only
	00000000)B	0,0,0,0 0,0,0,0,0,0,0	, 0, 0, 0, 0, 0, 0,	0,0,0,0,0
0004	=	10885 10886	; Default\$D	lisk	EQU 0004	4H ;Defau	lt disk in base page
		10887	;				
		10888	BOOT:		;Entered dim	rectly from	the BIOS JMP Vector
		10889					ferred here by the CP/M
		10890			; bootstra	p loader	
		10891			;		-1
		10892 10893					alize system routine uses the Initialize\$Stream
		10893					lared above
		10895				,	
1113	E3	10896	г	л		:Disab	le interrupts to prevent any
		10897	-				effects during initialization
1114	21A40F	10898	L	XI	H, Initializ		;HL -> data stream
	CD1903	10899	c	ALL	Output\$Byte		Output it to the specified
		10900					; ports
		10901					
111A	CDEE02	10902	C	ALL	General\$CIO	\$Initializa	tion ;Initialize character devices
		10903					
	21850F	10904		XI	H,Signon\$Me		;Display sign-on message on console
1120	CD5F02	10905		ALL	Display\$Mes	sage	
1100	CDDFOE	10906 10907	;	ALL	Patch\$CPM	• Make	necessary patches to CCP and BDOS
1123	CODFUE	10907	L L	·**	I ALCHIPUPPI		custom enhancements
		10908				, .01	eased ennancements
		10910				:Initi	alize M\$Disk
		10911				: If th	e M\$Disk directory has the
		10912					cial reserved file name "M\$disk"
		10913					th lowercase letters and marked
		10914				; SYS	and R/O), then the M\$Disk is
		10915				; ass	sumed to contain valid data.
		10916					e "M\$Disk" file is absent, the
		10917				; M\$E	lisk Directory entry is moved into
		10918				; the	M\$Disk image, and the remainder of directory set to OE5H.
		10919				; the	directory set to OE5H.
	0601	10920		1/1	B,1		t bank 1
1128	CDDDOB	10921	C	CALL	Select\$Bank	; whi	ch contains the M\$Disk directory
		10922					, if MtDiel divoctory anter averate
1100	210000	10923 10924		XI	н, о		if M\$Disk directory entry present address for first directory
	11F310	10924		_XI	D,M\$Disk\$Di		. GUDIESS FOR FIRST DIRECTORY
	0E20	10925		1VI	C, 32		h to compare
	ULZV	10928	n M\$Disk\$Te		0,02	,	an ee eempere
1133	1A	10928		DAX	D	;Get b	yte from initialized variable
1134		10929		MP	ň		are with M\$Disk image
	C24F11	10930		JNZ	M\$Disk\$Not\$;Match fails
1138	13	10931	1	INX	D		
1139	23	10932		INX	н		
113A	OD	10933		OCR	С		
	CA4111	10934		JZ	M\$Disk\$Setu		ytes match
113E	C33311	10935	· ·	JMP	M\$Disk\$Test		
		10936	J				
		10937	M\$Disk\$Se				
1141	218E10	10938	1	_XI	H,M\$Uisk\$Se	tup#Message	; ; Inform user
		10939	J Menialito		~ ~ •		
	ODEEAA	10940	M\$Disk\$Se				
1144	CD5F02	10941	C C	CALL	Display\$Mes	2996	
1147	AE	10942 10943		XRA	Δ	· Cat -	default disk drive to A:
	AF 320400	10943		STA	.A Default\$Dis		JEIGUIL DISK DEIVE LO DE
1148 114B		10944		EI	Selagi(#DIS		rrupts can now be enabled
1140		10945	1			, , , , , , , , , , , , , , , , , , , ,	tapts and now be analysed
1140	C36C02	10947		JMP	Enter\$CPM	:Go ir	nto CP/M
			•				

Figure 8-10. (Continued)

1049 HB01skN0456tur: LX1 D.0 :fNov HB01sk directory entry into 1152 217410 1093 LX1 H.HM51skND1rEntry ; H001skNDard 1157 1093 LX1 H.HM51skND1rEntry ; H001skNDard Forte blocks to move 1157 1093							
 144F 110000 10950 LXI D.0 PROM MEDISK directory entry into 1152 21730 10951 10953 CALL MoveR8 10954 10954 10955 10954 10955 10955 10055 10055 10055 10055 10055 10056 10057 1150 1150 10058 10058 10058 1150 10058 10058 10059 10059 10059 1150 10059 10059 10059 10050 1150 10058 10058 10059 10050 10050 1150 10050 10050 10050 1150 10050 10050 10050 1150 10050 10050 1150 1162 1163 1164 1164 1164 1165 1165<th></th><th>10949</th><th>MSDickSNotSctu</th><th>..</th><th></th><th></th><th></th>		10949	MSDickSNotSctu	. .			
1152 217310 10951 LTI H, HBD1skBD1*Entry ; HBD2 rd = Dyte blocks to move 1157 CDPB0A 10952 MI C.3278 ; DE -> next byte after HBD1sk directory 1157 CDPB0A 10955 ; DE -> next byte after HBD1sk directory ; entry in image 1150 CDPB0A 10955 WI A OESH ; Set up to do memory fill 1150 CDPB0A 10955 WI A OESH ; Set up to do memory fill 1150 CDPB0A 10956 WI C.((2 # 1024) - 32) / 8 ;TW allocation blocks 1151 CDPB0A 10964 CALL Howe88 iles Move88 to do fill operation 1164 CDPFC 10964 CALL Howe88 iles Move88 to do fill operation 1164 CDPFOA 10967 UP HDD1skBSetup8Done ; Output message and enter CP/H 1165 2CDF0A 10964 CALL Howe88 iles Move88 to do fill operation 1165 DF0A Last 8Tinitialized 8Dyter ; Passat 2Dytes for MBD1sk blocks to move 1165 DF0A CALL Howe88 iles address of last initialized byte	1145 110000					. M	
1155 00040 10952 MVI C.32/8 :Number of 8-byte blocks to move 1157 000700 10953 CALL Howses :set up to do memory fill 1154 8EE5 10957 MVI A,0E5H :set up to do memory fill 1155 8EE5 10957 MVI A,0E5H :set up to do memory fill 1155 8EE5 10957 MVI C.4(2 # 1024) - 32) / 8 ;Two allocation blocks 1156 12 10956 CALL Howses :les 32 bits for MDisk entry 1165 2000 10956 CALL Howses :les 32 bits for MDisk entry 1165 2001 10956 CALL Howses :les address of last initialized to do fill operation 1165 21001 10956 LXI H.HeDiskNot&Setup#Doe :lummy 1165 21001 10956 LXI H.HeDiskNot&Setup#Doe :lummy 1165 21001 10956 LXI H.HeDiskNot&Setup#Doe :lummy 1165 21001 10957 DB :lummy :lummy 1165 21001 10957 DB :lummy :lummy 1165 2001 10957 Hom cold boot initialization counter :lummy </td <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>							
1157 CDF80A 10535 CALL Howess $\frac{10753}{10755}$, $\frac{1}{10756}$, $\frac{1}{10757}$, $\frac{1}{10776}$, $\frac{1}{10777}$,					KSUITSEN	try	
<pre>iD= iD= iD= iD= iD= iD= iD= iD= iD= iD=</pre>							Number of 8-byte blocks to move;
10955	1157 CDF80A		CALL	Move\$8			
100%5111115021500%5WIA.OEMSet UP to do memory fill115110%5STAXDSet UPSet UP115210%5MOVL.ESet UPIse DE +1115400%5MOVL.ESet UPSet UP115400%5MOVL.ESet UPSet UP116000%6LXIH.VESet UPSet UP116210%64CALLHove88Set UPSet UP116521001010%64LXIH.MOIskNot4Setup3MessageSet UP116521001010%64LXIH.MOIskNot4Setup3MessageSet UP116521001010%64LXIH.MOIskNot4Setup3MessageSet UP116500%70J.M.BE 0Set UPSet UP116500%71LastSitisedSetUp3MessageSet UPSet UP116500%71LastSitisedSetUp3MessageSet UPSet UP116500%71LastSitisedSetUp3MessageSet UPSet UP116700%71LastSitisedSetUp3MessageSet UPSet UP116800%71LastSitisedSetUp3MessageSet UPSet UP116900%71LastSitisedSetUp3MessageSet UPSet UP116900%71LastSitisedSetUp3MessageSet UPSet UP116900%71LastSitisedSetUp3MessageSet UPSet UP116900%71LastSitisedSetUp3MessageSet UPSet UP11690		10954	;				
100%5111115021500%5WIA.OEMSet UP to do memory fill115110%5STAXDSet UPSet UP115210%5MOVL.ESet UPIse DE +1115400%5MOVL.ESet UPSet UP115400%5MOVL.ESet UPSet UP116000%6LXIH.VESet UPSet UP116210%64CALLHove88Set UPSet UP116521001010%64LXIH.MOIskNot4Setup3MessageSet UP116521001010%64LXIH.MOIskNot4Setup3MessageSet UP116521001010%64LXIH.MOIskNot4Setup3MessageSet UP116500%70J.M.BE 0Set UPSet UP116500%71LastSitisedSetUp3MessageSet UPSet UP116500%71LastSitisedSetUp3MessageSet UPSet UP116500%71LastSitisedSetUp3MessageSet UPSet UP116700%71LastSitisedSetUp3MessageSet UPSet UP116800%71LastSitisedSetUp3MessageSet UPSet UP116900%71LastSitisedSetUp3MessageSet UPSet UP116900%71LastSitisedSetUp3MessageSet UPSet UP116900%71LastSitisedSetUp3MessageSet UPSet UP116900%71LastSitisedSetUp3MessageSet UPSet UP11690		10955				• DE ->	> next byte after M\$Disk directory
115A 32E5 10957 MVI A,0E5H ;Set up to do memory fill 115C 12 10959 STAK D ;Set fill to do memory fill 115C 22 10950 MOV H, D ;Set fill to DE +1 115C 12 10950 MOV H, D ;Set fill to DE +1 115F 23 10961 MVI C,((2 * 1024) - 32) / 8 ;Two allocation blocks 1162 2CD#80A 10962 LXI H,wef8 ;Dest Say 2 bytes for blocks 1165 21CD10056 LXI H,wef8 ;Dest Say 2 bytes for blocks ;Dest Say 2 bytes for blocks 1165 21CD10056 LXI H,wef8 ;Dummy ;Dest Say 2 bytes for blocks 1165 21CD10056 LXI H,wef8 ;Dummy ;Dummy 1165 21CD10056 LXI H,wef8 ;Dummy ;Command sequences into the fill operation 10650 ; End of cold boot initialization code ; ;Command sequences into the fill operation 10677 ; DRG After\$Ength S128 ;This can be used to insert long ;comsole input pointer here 10677 ; ORG After\$Ength EOU 32 ;Must be binary number 2244 10976 ;DesBuffers! DS DisBuffers! 0020 = 10982 DesBuffer: DS DisBuff						,	ry in ince
1150 12 10958 STAK D istore first byte in "source" area 1150 62 10950 MOV L,E istore first byte in "source" area 1150 62 10950 MOV L,E istore first byte in "source" area 1160 0EFC 10963 MVI C,((2 * 1024) - 32) / 8 ;Two allocation blocks istore first byte in "source" area 1162 0EFC 10964 CALL Howes istore first byte in "source" area 1162 0EFC 10964 CAL Howes istore first byte in "source" area 1162 0EFC 10964 CAL Howes istore first byte in "source" area 1163 00 10965 LXI H, MDIskNotEstupMessage istore first byte in the source" area 1168 00 10970 LastBinitializedByte: ; Dummy istore address of last initialized byte 10971 Istore first byte in the source info the inter interve into the interve information the interve	1154 2005		MUT			, ent	
<pre>1150 62 10959 HOV H, D ; Set HL to DE *1 1157 23 10960 HOV L,E 1157 23 10964 INX H 1160 OEC 109963 INX H 1160 OEC 109963 CALL Hove\$8 ; less 2bytes for MPDisk entry 1162 CDF80A 10966 LXI H, HDDiskNot\$Stup\$Hessage 1165 21001 10966 LXI H, HDDiskNot\$Stup\$Hessage 1168 C3411 10967 , JHP HDDiskNot\$Stup\$Hessage 1168 C3411 10967 , JHP HDDiskNot\$Stup\$Hessage 1168 10977 Last\$Initialized\$Pte: ; <= address of last initialized byte 10977 , Last\$Initialized\$Pte: ; <= address of last initialized byte 10977 , Last\$Initialized\$Pte: ; <= address of last initialized byte 10977 , Last\$Initialized\$Pte: ; <= address of last initialized byte 10977 , Last\$Initialized\$Pte: ; <= address of last initialized byte 10977 , ecomand\$Stup\$Hest in the initialized in code 110975 , ORG After\$Disk\$Buffer ; Present location counter 10977 , ecomand\$Stup\$Hest in the initialized initialized in the initialized initialized in the initialized initialized</pre>							
<pre>115E 68 10%0 HOV LiE 115F 23 1160 OEFC 10%2 HVI C.((2 * 1024) - 32) / 8 ;Two allocation blocks 1162 CDF80A 10%6 CALL Hove89 ;Use Hove88 to do fill overalion 1165 21C01 10%6 CALL Hove89 ;Use Hove88 to do fill overalion 1165 21C01 10%6 CALL Hove89 ;Use Hove88 to do fill overalion 1165 21C01 10%6 CALL Hove89 ;Use Hove88 to do fill overalion 1165 21C01 10%6 CALL Hove89 ;Use Hove88 to do fill overalion 1165 21C01 10%6 CALL Hove89 ;Use Hove88 to do fill overalion 1165 21C01 10%6 CALL Hove89 ;Use Hove88 to do fill overalion 1165 21C01 10%6 CALL Hove89 ;Use Hove88 to do fill overalion 10%7 Last81nitialized89te: ;(~= address of last initialized byte 10%77 ; End of cold boot initialization code 10%77 Hulti@Command9Buffer: DS 128 ;This can be used to insert long 10%70 ; ORG After\$Length EOU 32 ;Hust be binary number 10%70 ; DOBSUffer\$Length EOU 32 ;Hust be binary number 2244 10%75 DOBSUffer\$Length EOU 32 ;Hust be binary number 2244 10%80 DOBSUffer\$Length EOU 32 ;Hust be binary number 2244 10%97 PI4User\$Length EOU 32 ;Hust be binary number 2266 10%97 PI4User\$Length EOU 32 ;Hust be binary number 2279 2 ; 2280 10000 j ; Disk BAUffer: DS 128 ;Disk directory buffer 10000 j ;Hust Work areas J 10000 j ;Hust Work area</pre>							
1157 23 1160 0EFC 1160 0EFC 1							;Set HL to DE +1
 1160 OEFC 10962 MVI C.((2 * 1024) - 32) / 8 ;Too allocation blocks 1 loss 32 bytes for MMDisk entry 1 loss 32 bytes for A for for B bytes for A for B byt				L,E			
1122 CDF804 10963 1142 CDF804 1048 For MBD1sk tentry 1142 CDF804 10965 1148 For MBD1sk tentry 1148 For MBD1sk tentry 1165 21001 10965 1148 For MBD1sk tentry 1148 For MBD1sk tentry 1165 21001 10966 1148 For MBD1sk tentry 1000000000000000000000000000000000000							
<pre>1020 EPG 32 Divest for MDDisk entry 102 CDF004 10965 103 CLL Moves 104 Divest to do fill operation 105 21001 10905 1090 10907 1090 10907 1090 1090 1090 10</pre>	1160 OEFC	10962	MVI	C,((2 *	(1024) -	· 32) /	8 :Two allocation blocks
1142 CDF80A 10964 CALL Move98 jUse Move88 to do fill operation 1145 CJLC010 10965 LXI H,MDDisk\$Not\$Setup\$Deme :0utput message and enter CP/M 1186 C3411 10967 JP MBDisk\$Sup\$Deme :0utput message and enter CP/M 1186 C3411 10977 DB 0 jDummy 1165 00 10977 Last\$Initialized\$Byte: :<= address of last initialized byte		10963					
1065 LXI H,MSDiskShot%Setup#Dene :Output message and enter CP/M 1165 2106 JMP MBDiskSSetup#Dene :Output message and enter CP/M 10680 10970 DB :Dummy :Cutput message and enter CP/M 10701 10970 DB :Dummy :Cutput message and enter CP/M 10771 :DB 0 :Dummy :Cutput message and enter CP/M 10772 :Edd of cold boot initialization code :Command sequences into the :Command sequences into the 10777 :Commod Sequences into the :Command sequences into the :Command sequences into the 10777 Multi%Command%Buffer: DS 128 :This can be used to insert long 10787 :Commod Sequences into the :Command sequences into the :Command sequences into the 10780 :Softer%Length EOU 32 :Must be binary number 0020 = :OSBUffer%Length EOU 32 :Must be binary number 02264 :OSBUffer%Length EOU 32 :Must be binary number 02264 :OSBUffer%Length DS D2%Buffer%Length :Command sequences into the	1162 CDE80A		CALL	Move\$8			
 1165 221010 1166 C24411 10966 J 10966 J 10967 DB 0 JDummy 10977 Last@Initialized%Byte: rimesage and enter CP/M 10977 Last@Initialized%Byte: rimesage and enter CP/M 10977 Last@Initialized%Byte: rimesage and enter CP/M 10977 Last@Initialized%Byte: rimesage 10977 Last@Initialized%Byte: rimesage 10977 Last@Initialized%Byte: rimesage 21A4 10977 ORG After%Disk%Buffer : Reset location counter 10977 J ORG After%Disk%Buffer : reset location counter 10977 Last@Initialized%Byte: rimesage 21A4 10977 ORG After%Disk%Buffer : reset location counter 10978 Lost@Initialized%Dyte: reset location counter 10979 Lost@Initialized%Dyte: reset location counter 10977 Lost@Initialized%Dyte: reset location counter 10978 Lost@Initialized%Dyte: reset location counter 10979 Lost@Initialized%Dyte: reset location counter 10976 Lost@Initialized%Dyte: reset location counter 10977 Lost@Initialized%Dyte: reset location counter 10978 Lost@Initialized%Dyte: reset location counter 10979 Lost@Initialized%Dyte: reset location counter 10970 Lost@Initialized%Dyte: reset location counter 10971 Lost@Initialized%Dyte: reset location counter 10972 Lost@Initialized%Dyte: reset location counter 10973 Lost@Initialized%Dyte: reset location counter 10973 Lost@Initialized%Dyte: reset location counter 10974 Lost@Initialized%Dyte: reset location counter 10975 Lost@Initialized%Dyte: reset location 10980 Lost@I							yose morero to do rite operation
<pre>1168 C34411 10967 UMP M\$Disk\$Setup\$Done :Dutput message and enter CP/M 10968 ; 10969 ; 1080 DB 0 :Dummy 10971 LastBinitializedByte: ::Dummy 10971 LastBinitializedByte: :::C== address of last initialized byte 10972 ; 10973 ; End of cold boot initialization code 10974 ; End of cold boot initialization code 10975 ; ORG After\$Disk\$Buffer : Reset location counter 10976 ; 10977 ; ORG After\$Disk\$Buffer :Reset location counter 10977 ; Command&Buffer: DS 128 ;This can be used to insert long 10978 ; Command&Buffer: DS 128 ;This can be used to insert long 10978 ; Command SBuffer: DS 20 ;Flust be binary number 10979 ; Command SBuffer*Length 10981 ; DO\$Buffer*Length 10983 ; DO\$Buffer*Length 10983 ; DO\$Buffer*Length 10984 ; DO\$Buffer*Length 10983 ; DO\$Buffer*Length 10983 ; DO\$Buffer*Length 10984 ; DO\$Buffer*Length 10984 ; DO\$Buffer*Length 10989 ; DI\$Buffer*Length 10989 ; DI\$Buffer*Length</pre>	1145 210010		1.77		LENGIECO	+	
<pre>1168 00 10960 ; 10970 BB 0 ;Dummy 10971 Last%Initialized%Pyte: :<= address of last initialized byte 10972 ; 10972 ; 10973 CR0 After%Disk%Buffer ;Reset location counter 10975 ;OR0 After%Disk%Buffer ;Reset location counter 10976 ; 21A4 10977 Multi%Command%Buffer: DS 128 ;This can be used to insert long 10979 ; Command sequences into the 10976 ; 10981 ;DSDUffer%Length EQU 32 ;Must be binary number 2224 10983 DO#Buffer*Length EQU 32 ;Must be binary number 2244 10986 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10986 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10988 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10989 DI\$Buffer*Length EQU 32 ;Must be binary number 2244 10999 ; 2286 10997 FI\$User\$Hu: DS 2 ;Save area for user's stack pointer 10999 ; 1002 i Disk work areas 10997 ; 2380 11002 i \$\must her pointer 10999 ; 2380 11002 i Disk work areas 10002 i Disk Work areas 10003 ; Disk Work areas 10004 ; 1003 ;</pre>							
<pre>1168 00 10969 ; DB 0 ;Dummy 10970 LastEinitializedByte: ;C== address of last initialized byte 10972 ; 10972 ; 10977 ; End of cold boot initialization code 21A4 10977 ; ORO After\$Disk#Buffer ;Reset location counter 10978 ; Ormond #Buffer: DS 128 ;This can be used to insert long 10979 ; The forced input pointer here 10980 ; the forced input pointer here 10981 ; ORSO 108Buffer: DS DOBBuffer\$Length 10984 ; 0020 = 10982 DisBuffer\$Length EGU 32 ;Hust be binary number 2244 10983 DisBuffer: DS DOBBuffer\$Length 10987 D28Buffer: DS DD2Buffer\$Length 10989 i D28Buffer: DS D28Buffer\$Length 10989 i Orse area for user's stack pointer 10981 i Data areas for the character drivers 10992 i vers's stack area 10993 P18User\$HL: DS 2 ;Save area for user's stack pointer 10995 i vers's stack area 10996 i vers's stack area 10996 i vers's stack area 10997 i vers's stack area 10097 i vers's stack area</pre>	1168 034411		e	M#U15K\$	iSetup\$Do	ne	;Uutput message and enter CP/M
1168 00 10970 DB 0 ;Dummy 10971 Last&finitializedSyte: ;<=====================							
<pre>1071 LastSInitializedtByte: ;<= address of last initialized byte 1077 ; 1077 ; 1077 ; 21A4 1077 0RG After*Disk#Buffer ;Reset location counter 1077 Multi#Command#Buffer: DS 128 ;This can be used to insert long 1077 Multi#Command#Buffer: DS 128 ;Thust be binary number 1077 1078 DO\$Buffer*Length EOU 32 ;Must be binary number 2244 10785 DO\$Buffer*Length EOU 32 ;Must be binary number 2244 10785 DO\$Buffer*Length EOU 32 ;Must be binary number 2244 10786 DI#Buffer*Length EOU 32 ;Must be binary number 2244 10787 DO\$Buffer*Length EOU 32 ;Must be binary number 2244 10788 DI#Buffer*Length EOU 32 ;Must be binary number 2244 10789 Do\$Buffer*Length EOU 32 ;Must be binary number 2244 10789 Di#Buffer*Length EOU 32 ;Must be binary number 2244 10789 Di#Buffer*Length EOU 32 ;Must be binary number 2244 10799 potential potential in the interrupt occurs 2248 10799 potential intervent occurs 2280 10797 i Data areas for the character drivers 10797 i user's stack area 10797 i change of disketter to read-only status. 2380 11011 Diskt#Auffer: DS 128 ;Disk directory buffer 11007 i change of disketter to read-only status. 2380 11012 Diskt#Auffer: DS 16 ;D: 2400 11012 Diskt#Auffer: DS 16 ;D: 2400 11013 Diskt@tworkareas: DS 16 ;D: 2400 11014 Diskt@tworkareas: DS 16 ;D: 2400 11015 i Disk user for use of an only status. 2380 11013 Diskt@tworkareas: DS 16 ;D: 2400 11014 Diskt@tworkareas: DS 16 ;D: 2400 11015 Diskt@tworkareas: DS 16 ;D: 2400 11015 i Diskt@tworkareas: DS 16 ;D: 2400 11015 i Diskt@tworkareas: DS 16 ;D: 2400 11016 i Diskt@tworkareas: DS 16 ;D: 2400 11017 i These are used by the EDOS to maintain a bit map of 11020 ; which allocation vectors 11016 ; 11015 ; 110</pre>							
<pre>10772 ; End of cold boot initialization code 10774 ; 10775 ; ORG After&DiskBuffer : Reset location counter 10776 ; 21A4 10777 ; Hulti@Command%Buffer: DS 128 ; This can be used to insert long 10778 ; command sequences into the 2 console input stream by setting 2 command sequences into the 2 command sequences into the intermediance into the 2 command sequences into the intermediance into the interviewed interviewed interviewed into the interviewed interviewed into the interviewed interviewed interviewed interviewed interviewed into the interviewed interviewed interviewed into the interviewed interviewed interviewed</pre>	116B 00		DB	0		; Dummy	/
<pre>10772 ; End of cold boot initialization code 10774 ; 10775 ; ORG After&DiskBuffer : Reset location counter 10776 ; 21A4 10777 ; Hulti@Command%Buffer: DS 128 ; This can be used to insert long 10778 ; command sequences into the 2 console input stream by setting 2 command sequences into the 2 command sequences into the intermediance into the 2 command sequences into the intermediance into the interviewed interviewed interviewed into the interviewed interviewed into the interviewed interviewed interviewed interviewed interviewed into the interviewed interviewed interviewed into the interviewed interviewed interviewed</pre>		10971	Last\$Initialize	d\$Byte:			
10773End of cold boot initialization code21A410775ORGAfter&Disk&Buffer: FReset location counter21A410777Multi@Command\$Buffer:DS128:This can be used to insert long10778: command sequences into the: command sequences into the10779: command sequences into the10779: command sequences into the10781: command sequences into the10792: command sequences into the10793: the forced input pointer here10781: comsol power10782: comsol power10783: the forced input pointer here10784: comsol power10785: comsol power0020 =: input pointer0020 =: input pointer10786: input power2224: input power10787: input power2224: input power0020 =: input power0020 =: input power0020 =: input power10787: input power: input power </td <td></td> <td></td> <td></td> <td>-</td> <td></td> <td></td> <td>,</td>				-			,
 21A4 10974 10975 0RG After\$Disk\$Buffer ;Reset location counter ;0976 10977 10977 10977 10978 10978 10978 10978 10979 10979 10979 10979 10979 10979 10970 10980 10990 10991 10980 10991 10980 10991 10980 10991 10980 10991 10980 10991 10980 10991 10991<td></td><td></td><td></td><td>cold boo</td><td>nt initi=</td><td>lizatio</td><td>n code</td>				cold boo	nt initi=	lizatio	n code
21A4109750RGAfter\$Disk\$Buffer:Reset location counter21A410977Multi\$Commad\$Buffer:DS128;This can be used to insert long10978i commad \$Buffer:DS128; commad \$euences into the10979i commad \$Euences into the; commad \$euences into the10981i 0080uffer\$LengthEQU32;Must be binary number222410983D08Buffer\$LengthEQU32;Must be binary number0020 =10981D1\$Buffer\$LengthEQU32;Must be binary number224410986D1\$Buffer\$LengthEQU32;Must be binary number10970iDosD2\$Buffer\$LengthEQU320020 =10988D2\$Buffer\$LengthEQU32;Must be binary number226410998D2\$Buffer\$LengthEQU32;Must be binary number228410993D2\$Buffer\$LengthEQU32;Must be binary number228410995D2\$Buffer\$LengthEQU32;Must be binary number228610995P1\$Buser\$Stack:Disterstructorsisterstructors228010996P1\$User\$Stack:Disterstructorsisterstructors10991iData areas for the character driversisterstructorsisterstructors228010995P1\$User\$Stack:Disterstructorsisterstructors10997iDisterstructorsisterstructorsisterstructors10998i<							
1097610977Multi\$Command\$Buffer:DS128:This can be used to insert long ; command sequences into the ; comsole input stream by setting ; the forced input stream by setting ; the forced input setting ; the forced input setting0020 =10982D08Buffer\$LengthEQU32;Must be binary number1098410984FriDSD08Buffer\$Length2140020 =10985D18Buffer\$LengthEQU32;Must be binary number224410986D18Buffer\$LengthEQU32;Must be binary number224410986D28Buffer\$LengthEQU32;Must be binary number224410986D28Buffer\$LengthEQU32;Must be binary number224410986D28Buffer\$LengthEQU32;Must be binary number10997;Data areas for the character drivers110998if differDS2 iStorage area for user's stack pointer10999;if differDS4010997PI8User\$HL:DS2 iStorage area for user's stack read10998;if differDS12810999;if differDS128228011000p; corry\$Buffer;DS12810999;if differDS128230011001ifif differ11003;Disk work areas1100411004;If sit differDS230011010Disk&Workarea:DS <t< td=""><td>2164</td><td></td><td></td><td>A</td><td>1</td><td></td><td>Posst logation counter</td></t<>	2164			A	1		Posst logation counter
21A410977 10978Multi#Command#Buffer: 10979DS128 ite command sequences into the ite command sequences into the ite command s	2107		UKO	niter#D	×15K≠DUIT	er	Aneset location counter
10978: command sequences into the10979; comsole input stream by setting0020 =10982DoBsUffer%LengthEOU3210981;the forced input pointer here10981;DoBsUffer%LengthEOU222410983DoBsUffer%LengthEOU3210984;IsBuffer%LengthEOU32224410986DisBuffer%LengthEOU3210987;DisBuffer%LengthEOU32224410986DisBuffer%LengthEOU3210988D28Buffer%LengthEOU32224410986DisBuffer%LengthEOU224410988D28Buffer%LengthEOU224410988D28Buffer%LengthEOU226410997;Data areas for the character drivers10997;Data areas for the character drivers10998;stack area for user's stack pointer10999;ioutines the jointer is vice10996;ioutines the jointer is stack area10997PisUser\$HL:DS128228011000Director\$Buffer:DS11011;iuser's stack area11023;Hsbisk11044;iuser's stack area11055;Disk work areas11060;iuser's stack area11061;iuser's stack area11062;bisk work areas11063;iuser's stack area<			7				<u>-</u>
10979; consle input stream by setting ; the forced input points here0020 =10981;222410983D05Buffer%LengthEOU32 ; Must be binary number222410985D18BufferDSD18Buffer%Length0020 =10985D18Buffer%LengthEOU32 ; Must be binary number224410986D18Buffer%LengthEOU32 ; Must be binary number224410987jD28BufferDS10987jD28BufferDSD28Buffer%Length10990 ;1Data areas for the character drivers10991 jData areas for the character drivers10992 ;jStorage area for user's stack pointer10993 p19User\$Stack:DSjStack area for user's stack pointer10994 ;jrotinejrotine228610995P19User\$HL:DS2JSave area for user's stack pointer10999 ;jrotinejrotinej228011000Directory\$Buffer:DS128 ; Disk directory buffer11001 ;jDisk vork areasj11003 ;iDisk vork areasj11004 ;isuchastreatDS228011012Disk vork areasDS11003 ;iDisk vork areasS11004 ;Disk vork areasDS32 ; A:230011012Disk vork areasDS11003 ;ichange of diskettes to read-only status. <td>21A4</td> <td></td> <td>Mult1%Command%E</td> <td>utter:</td> <td>DS</td> <td>128</td> <td></td>	21A4		Mult1%Command%E	utter:	DS	128	
10980 1090 ; the forced input pointer here 1092 ; the forced input pointer here 10982 D098Uffer%Length EOU 32 ; Must be binary number 10984 ; 10984 ; 10984 ; 10985 D198Uffer%Length EOU 32 ; Must be binary number 2244 10986 D198Uffer%Length EOU 32 ; Must be binary number 2256 10989 D298Uffer%Length EOU 32 ; Must be binary number 2264 10989 D298Uffer%Length EOU 32 ; Must be binary number 2286 10997 ; 2286 10997 PISUser%Stack: DS 2 ; Storage area for user's stack pointer 10991 ; Data areas for the character drivers 10992 ; 2286 10995 PISUser%Stack: DS 2 ; Storage area for user's stack pointer 10994 ; user's HL 2286 10995 PISUser%Stack: DS 2 ; Storage area for use by interrunt service 10996 ; user's stack area for use by interrunt service 10997 ; 2280 11000 Directory%Buffer: DS 128 ;Disk directory buffer 11001 ; 2330 11002 M%Disk\$Buffer: DS 128 ;Disk directory buffer 11004 ; 11005 ; Disk work areas 11006 ; 11006 ; 11006 ; 11006 ; 2380 11011 Disk\$Af\$Work areas 11006 ; 2380 11011 Disk\$Af\$Work areas DS 32 ; A: 2380 11011 Disk\$Af\$Work areas DS 32 ; A: 2380 11012 Disk\$Af\$Work areas DS 32 ; A: 2380 11011 Disk\$Af\$Work areas DS 32 ; A: 2380 11012 Disk\$Af\$Work areas DS 32 ; A: 2380 11012 Disk\$Af\$Work areas DS 32 ; A: 2380 11012 Disk\$Af\$Work areas DS 32 ; A: 2380 11014 Disk\$Af\$Work areas DS 32 ; B: 2380 11015 ; 11016 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11016 ; 11016 ; 11016 ; 11016 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11016 ; 11016 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11016 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks/80+1.							
10981jEOU32jMust be binary number222410983D098uffer%LengthEOU32jMust be binary number224410985D198uffer%LengthEOU32jMust be binary number224410986D198uffer%LengthEOU32jMust be binary number224410986D198uffer%LengthEOU32jMust be binary number226410989D298ufferDSD198uffer%LengthEOU10991jData areas for the character drivers10991jData areas for the character drivers10992jDs0228410993P19User\$Stack:DS10995jDs0228610995P19User\$HL:DS228011000Directory\$Uffer:DS11000Directory\$Uffer:DS12811001jDisk work areas11002the binary dust areas11004jIntermediary buffer for11004jDisk Work areas11005jDisk Work areas11006jStack area11007jThese are used by the BDOS to detect any unexpected11008 <td< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td>; console input stream by setting</td></td<>							; console input stream by setting
0020 = 10982 D00Buffertlength EOU 32 PMust be binary number 0224 10984 ; D0Buffertlength EOU 32 ;Must be binary number 0220 = 10985 D15Buffertlength EOU 32 ;Must be binary number 2234 10986 D15Buffertlength EOU 32 ;Must be binary number 2236 10988 D25Buffertlength EOU 32 ;Must be binary number 2236 10989 D25Buffertlength EOU 32 ;Must be binary number 2236 10997 ; Data areas for the character drivers ; ;Must be binary number 2286 10997 ; Data areas for the character drivers ; ;Must be binary number 2286 10997 ; Data areas for the character drivers ; ;Must be binary number 2286 10997 ; Data areas for the character drivers ; ;Must be binary number 2286 10997 ;Plsusr\$\$ Plsusr\$\$;Sack area for user's HL ;Sa		10980					; the forced input pointer here
222410983DOSBuffer:DSDOSBuffer\$Length10985D1\$Buffer\$LengthEOU32:Nust be binary number224410985D1\$Buffer\$LengthEOU32:Nust be binary number224410987:D2\$Buffer\$LengthEOU32:Nust be binary number226410989D2\$Buffer\$LengthEOU32:Nust be binary number226410999iData areas for the character driversi10971:Data areas for the character driversi10973P1\$User\$Stack:DS2:Storage area for user's stack pointer228410975P1\$User\$Stack:DS2:Storage area for user's HL228410975P1\$User\$HL:DS2:Storage area for user's HL228610976P1\$User\$HL:DS2:Storage area for user's HL228810976P1\$Stack::routines to avoid overflowing the10979:istack area:storage for user's stack228011000Director\$Buffer:DS12811004::istskifer:DS2300::Disk work areas11004::.storage for user's istack2300::::2300:::1001::::::::::::::::::	1	10981	;				
222410983DOSBuffer:DSDOSBuffer\$Length10985D1\$Buffer\$LengthEOU32:Nust be binary number224410985D1\$Buffer\$LengthEOU32:Nust be binary number224410987:D2\$Buffer\$LengthEOU32:Nust be binary number226410989D2\$Buffer\$LengthEOU32:Nust be binary number226410999iData areas for the character driversi10971:Data areas for the character driversi10973P1\$User\$Stack:DS2:Storage area for user's stack pointer228410975P1\$User\$Stack:DS2:Storage area for user's HL228410975P1\$User\$HL:DS2:Storage area for user's HL228610976P1\$User\$HL:DS2:Storage area for user's HL228810976P1\$Stack::routines to avoid overflowing the10979:istack area:storage for user's stack228011000Director\$Buffer:DS12811004::istskifer:DS2300::Disk work areas11004::.storage for user's istack2300::::2300:::1001::::::::::::::::::	0020 =	10982	DO\$Buffer\$Lengt	h	EQU	32	:Must be binary number
10984;EOU32; Must be binary number224410986D1\$Buffer\$LengthEOU32; Must be binary number224410986D2\$Buffer\$LengthEOU32; Must be binary number226410988D2\$Buffer\$LengthEOU32; Must be binary number226410980p2\$BufferD2\$Buffer\$Length10970;10991;Data areas for the character drivers10972;10992;Data areas for the character drivers1097210993P1\$User\$Lack:DS2; Storage area for user's stack pointer228610995P1\$User\$HL:DS2; Stack area for user's HL228610995P1\$User\$HL:DS128; Disk directory buffer10994;rrutines to avoid overflowing the10995;;rottick area10996;;rutines to avoid overflowing the10997;jiskoffer:DS128228011000Directory\$Buffer:DS12811001;;isk work areas11005;Disk work areas11006;isk work areas11007;These are used by the BDOS to detect any unexpected11006;isk a changed diskette to read-only status.11007;These are used by the BDOS to maintainally st11009;such a changed diskette to read-only status.11009;such a changed	2224	10983			DO\$Buff	ersleng	ath
0020 =10985DisBuffersLengthEOU32:Nust be binary number22441098710020 =10988D28BuffersLengthEOU32:Must be binary number224410987D28BuffersLengthEOU32:Must be binary number224410987D28BuffersLengthEOU32:Must be binary number228410990;D28BuffersLengthEOU32:Must be binary number228410990;Data areas for the character drivers:							
224410%6D1%Buffer:DSD1%Buffer%Length10020 =10%88D2%BufferfsLengthEOU32:Must be binary number226410%87D2%Buffer:DSD2%Buffer%Length10%90;Data areas for the character drivers10%91;Data areas for the character drivers10%92;Data areas for the character drivers10%92;Data areas for the character drivers10%92;when an interrupt occurs228410%95P1%User%HL:DS228510%96DS4010%97P1%Stack:;vulnes to avoid overflowing the10%97p1%Stack:;user's stack area10%97jisterter:DS128228011000irectory%Buffer:DS11001;isterter:DS11003;Disk work areas11004;isterter:11005;Disk work areas11006;change of diskettes. The BDCS to detect any unexpected11007;These are used by the BDCS to detect any unexpected11009;such a changed diskette to read-only status.11009;such a changed diskettes.228011011Disk\$	0020 -		7 DisDuffassel anat	h	FOU	22	. Must be binned number
10987:EOU32:Must be binary number226410988D2\$Buffer:DSD2\$Buffer\$LengthEOU32:Must be binary number226410989D2\$Buffer:DSD2\$Buffer\$Lengthitem an interrupt occurs10992;Data areas for the character driversitem an interrupt occurs10994;Save area for user's HL228610995P1#User\$HL:DS2; Save area for use by interrupt service10994item an interrupt occursitem an interrupt occurs228610995P1#User\$HL:DS2; Save area for use by interrupt service10997item an interrupt occursitem an interrupt occursitem an interrupt occurs228010996item an interrupt occursitem an interrupt occurs10009item an interrupt occursitem an interrupt occurs1001item an interrupt occursitem an interrupt occurs228011000Directory\$Buffer:DS12811001item an interrupt occursitem an interrupt occurs11002M\$Disk\$Buffer:DS12811003item an interrupt occursitem an interrupt occurs11004item an interrupt occursitem an interrupt occurs11005item an interrupt occursitem an interrupt occurs11006item an interrupt occursitem an interrupt occurs11007item an interrupt occursitem an interrupt occurs11008item an interrupt occursitem an interrupt occ							
0020 =10988 10990D2\$Buffer\$Length D3 10990EOU 3232:Must be binary number number D2\$Buffer\$Length226410990 10991;Data areas for the character drivers 10991;228410993P1\$User\$Stack: DS2; Storage area for user's stack pointer ; when an interrupt occurs228410993P1\$User\$Stack: DS2; Storage area for user's stack pointer ; when an interrupt occurs228610995P1\$User\$HL: DS2; Save area for user's HL228810996DS40; Stack area for user of user's HL228010996;routines to avoid overflowing the ; routines to avoid overflowing the ; routines to avoid overflowing the ; user's stack area10097p1\$Stack:DS128; Disk directory buffer230011002M*Disk*Buffer:DS128; Intermediary buffer for ; M*Disk11004;11005;Disk work areas:11005;Disk work areas::11006;these are used by the BDOS to detect any unexpected11008; change of diskette to read-only status.11009;such a changed diskette to read-only status.23B011011Disk#A\$Workarea:DS23B011012Disk#A\$Workarea:DS23B011013Disk#A\$Workarea:DS240011014Disk#A\$Workarea:DS11015;1101511016;11017;	2244		DI\$Buffer:	05	DI#BUTT	eraLeng) în
226410989D2\$Buffer:DSD2\$Buffer\$Length10990;Data areas for the character drivers10991;Data areas for the character drivers228410992;10994;Storage area for user's stack pointer10995P1\$User\$HL:DS2228610996DS4010997P1\$Stack:;routines to avoid overflowing the10998;user's stack areafor use by interrupt service10999;22801000Directory\$Buffer:DS1281001;23301100M*Disk*Buffer:DS12811001;11003;Disk work areas.11004;11005;Disk work areas11006;11007;These are used by the BDOS to detect any unexpected11008;11009;such a changed diskette to read-only status.11010;228011011Disk*A\$Workarea:DS237011012Disk*A\$Workarea:DS238011014Disk*A\$Workarea:DS240011015;11015;.11016;.11017;Disk allocation vectors11018;.11019;These are used by t			;				
10990;10991;22841099210992;22841099310994;22861099510995PI\$User\$HL:22881099610996DS22801099710997PI\$Stack:10998;10999;2280100010001;23001100211002#BDisk\$Buffer:1003;11004;11005;11005;11006;11006;11007;11008;11009;11009;11004;11005;11005;11006;11007;11008;11009;11009;11010j11021011110210121103101411015110161101711018110181101911014110191101911019110191101911019110191101211020110211102211022110231102311023							;Must be binary number
10990;10991;22841099210992;22841099310994;22861099510995PI\$User\$HL:22881099610996DS22801099710997PI\$Stack:10998;10999;2280100010001;23001100211002#BDisk\$Buffer:1003;11004;11005;11005;11006;11006;11007;11008;11009;11009;11004;11005;11005;11006;11007;11008;11009;11009;11010j11021011110210121103101411015110161101711018110181101911014110191101911019110191101911019110191101211020110211102211022110231102311023	2264	10989	D2\$Buffer:	DS	D2\$Buff	er\$Leng	ath
10992 10993PI\$User\$Stack:DS2:Storage area for user's stack pointer ; when an interrupt occurs228610995PI\$User\$HL:DS2;Save area for user's HL228810996DS40;Stack area for user's HL228910997PI\$Stack:; routines to avoid overflowing the ; user's stack area10997j::228011000Directory\$Buffer:DS1281001;::233011002M\$Disk\$Buffer:DS12811004;:::11005:Disk work areas:11006::::11007:These are used by the BDOS to detect any unexpected ito0811007:These are used by the BDOS will automatically set ito0911009:::238011011Disk\$A\$Workarea:DS128011012Disk\$A\$Workarea:DS238011011Disk\$A\$Workarea:DS11010:::238011011Disk\$A\$Workarea:DS11016:::240011014Disk\$A\$Workarea:DS11015:::11016::11017:Disk allocation vectors11018::11019:These are used by the BDOS to maintain a bit map of11020:::11017 <td< td=""><td></td><td>10990</td><td>;</td><td></td><td></td><td></td><td></td></td<>		10990	;				
228410993 10994PI\$User\$Stack: DS2; Storage area for user's stack pointer ; when an interrupt occurs228610995PI\$User\$HL:DS2; Save area for user's HL228810996DS40; Stack area for user's HL228910997PI\$Stack:; routines to avoid overflowing the ; user's stack area interrupt service10997PI\$Stack:; routines to avoid overflowing the ; user's stack area10997j;::228011000Directory\$Buffer:DS12811001;::233011002M\$Disk\$Buffer:DS12811004;:::11005;Disk work areas11006;::11007;These are used by the BDOS to detect any unexpected inform11008;::11009;such a changed diskette to read-only status.11010:::23B011012Disk\$A\$Horkarea:DS23D011012Disk\$A\$Horkarea:DS240011014Disk\$A\$Horkarea:DS240011015;11017;Disk allocation vectors11018::11019;These are used by the BDOS to maintain a bit map of :11010::2400::11017:Disk allocation vectors11018::11019 </td <td></td> <td>10991</td> <td>: Data ar</td> <td>eas for</td> <td>the char</td> <td>acter d</td> <td>irivers</td>		10991	: Data ar	eas for	the char	acter d	irivers
228410993 10994PI\$User\$Stack: DS2; Storage area for user's stack pointer ; when an interrupt occurs228610995PI\$User\$HL:DS2; Save area for user's HL228810996DS40; Stack area for user's HL228910997PI\$Stack:; routines to avoid overflowing the ; user's stack area interrupt service10997PI\$Stack:; routines to avoid overflowing the ; user's stack area10997j;::228011000Directory\$Buffer:DS12811001;::233011002M\$Disk\$Buffer:DS12811004;:::11005;Disk work areas11006;::11007;These are used by the BDOS to detect any unexpected inform11008;::11009;such a changed diskette to read-only status.11010:::23B011012Disk\$A\$Horkarea:DS23D011012Disk\$A\$Horkarea:DS240011014Disk\$A\$Horkarea:DS240011015;11017;Disk allocation vectors11018::11019;These are used by the BDOS to maintain a bit map of :11010::2400::11017:Disk allocation vectors11018::11019 </td <td></td> <td>10992</td> <td></td> <td></td> <td></td> <td></td> <td></td>		10992					
10994i when an interrupt occurs228610995PI\$User\$HL: DS2 ;Save area for user's HL228810996DS40 ;Stack area for use by interrupt service10997PI\$Stack:: routines to avoid overflowing the10998: user's stack area10999:100011000Directory\$Buffer:DS123011002M\$Disk\$Buffer:DS123011002M\$Disk\$Buffer:DS123011002M\$Disk\$Buffer:DS123011005;Disk work areas11006;11007;These are used by the BDOS to detect any unexpected11008;change of diskettes. The BDOS will automatically set11009;such a changed diskette to read-only status.11010;1012238011011Disk\$A\$HWorkarea:DS238011012Disk\$A\$HWorkarea:DS238011014Disk\$\$A\$HWorkarea:DS240011014Disk\$\$D\$HWorkarea:DS11015;.11016;11015;11016;11017;11018;11019;11019;11014;11015;11015;11016;11017;1101811019;11019;11019;1101911019 </td <td>2284</td> <td></td> <td>PI\$IIser\$Stack:</td> <td>ns</td> <td>2</td> <td>:Stora</td> <td>oe area for user's stark pointer</td>	2284		PI\$IIser\$Stack:	ns	2	:Stora	oe area for user's stark pointer
228610995PI\$User\$HL:DS2;Save area for user's HL228810996DS40;Stack area for use by interrupt service10997itestack:; rottines to avoid overflowing the10998; user's stack area10999itestack:DS11000Directory\$Buffer:DS11001;23301100211003;11004;11005:11005;11006;11006;11007:11008:11009:238011010:23801101123801101211013238011014110152380110151101511016110171101811019 <t< td=""><td></td><td></td><td>· · · · · · · · · · · · · · · · · · ·</td><td></td><td>-</td><td></td><td></td></t<>			· · · · · · · · · · · · · · · · · · ·		-		
228810996DS40; Stack area for use by interrupt service10997PI\$Stack;; routines to avoid overflowing the10998; user's stack area10999;22801100011001;23301100211003; BS11004;11004;11005: Disk work areas11006;11007: These are used by the BDOS to detect any unexpected11008: change of diskettes. The BDOS will automatically set11009; such a changed diskette to read-only status.11009: such a changed diskette to read-only status.238011011Disk\$\$BWorkarea:DS238011012Disk\$\$D\$Workarea:DS238011013Disk\$\$D\$Workarea:DS24001104;11015:11016:11017:11018:11019:11019:11019:110121101311014::11015:::::::::::::::::::::::::::	2284			ne	2		
<pre>10997 PI\$Stack: ; routines to avoid overflowing the 10998 ; user's stack area 10999 ; 2280 11000 Directory\$Buffer: DS 128 ;Disk directory buffer 11001 ; 2330 11002 M*Disk\$Buffer: DS 128 ;Intermediary buffer for 11003 ; M*Disk 11004 ; 11005 ; Disk work areas 11006 ; 11007 ; These are used by the BDOS to detect any unexpected 11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 11010 ; 23B0 11011 Disk\$A\$Workarea: DS 32 ; A: 23B0 11012 Disk\$E\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; D: 11015 ; 11015 ; 11016 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1.</pre>			F1#Oser#AL:				
10998; user's stack area10999;22B01100011001;233011002M*Disk*Buffer:DS128;Intermediary buffer for11003;11004;11005Disk work areas11006;11007;11008;11009;11009;11009;11009;11009;11009;11009;11009;11009;11009;11009;11010;11010;23B01101111011Disk*A\$Workarea:23101101211012Disk*A\$Workarea:23101101311014Disk*A\$Workarea:23101101311015;11016;11017;110181101811019:11019:11019:11019:1101811019110191101211021110221102311023	2288			05	40		
10999;22B011000Directory\$Buffer:DS128;Disk directory buffer233011002M\$Disk\$Buffer:DS128;Intermediary buffer for233011003;itermediary buffer for11004;itermediary buffer for11005;Disk work areas11006;itermediary buffer for11007;These are used by the BDOS to detect any unexpected11008;change of diskettes. The BDOS will automatically set11009;such a changed diskette to read-only status.11010;101223B011011Disk\$4\$Workarea:DS23F011012Disk\$4\$Workarea:DS23F011013Disk\$4\$Workarea:DS240011014Disk\$4\$Workarea:DS11015;1101511015;These are used by the BDOS to maintain a bit map of11019;These are used for eight allocation blocks, hence the11021;One byte is used for eight allocation blocks, hence the11022;expression of the form (allocation blocks/8)+1.			PI\$Stack:				
22B01100Directory\$Buffer:DS128;Disk directory buffer233011002M\$Disk\$Buffer:DS128;Intermediary buffer for11003;	1					; use	er's stack area
<pre>11001 ; 2330 11002 M*Disk*Buffer: DS 128 ;Intermediary buffer for 11003 ; M*Disk 11004 ; 11005 ; Disk work areas 11006 ; 11007 ; These are used by the BDOS to detect any unexpected 11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 11010 ; 23B0 11011 Disk*A\$Workarea: DS 32 ; A: 23D0 11012 Disk*B\$Workarea: DS 16 ; C: 2400 11014 Disk*S\$Workarea: DS 16 ; C: 2400 11015 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1.</pre>			;				
 2330 11002 M*Disk*Buffer: DS 128 ;Intermediary buffer for 11003 ; M*Disk 11004 ; 11005 ; Disk work areas 11006 ; 11007 ; These are used by the BDOS to detect any unexpected 11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 23B0 11011 Disk*A*Workarea: DS 32 ; A: 23B0 11012 Disk*B*Workarea: DS 32 ; B: 23F0 11013 Disk*G*Workarea: DS 16 ; C: 2400 11014 Disk*D*Workarea: DS 16 ; D: 11015 ; 11015 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 	22B0		Directory\$Buffe	r:	DS	128	;Disk directory buffer
<pre>11003 ; M*Disk 11004 ; 11005 ; Disk work areas 11006 ; 11007 ; These are used by the BDOS to detect any unexpected 11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 11010 ; 23B0 11011 Disk*A\$Workarea: DS 32 ; A: 23B0 11012 Disk*B\$Workarea: DS 32 ; B: 23F0 11013 Disk*C*Workarea: DS 16 ; C: 2400 11014 Disk*D\$Workarea: DS 16 ; C: 2400 11014 Disk*D\$Workarea: DS 16 ; C: 11015 ; 11016 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1.</pre>	1	11001	;				
<pre>11003 ; M*Disk 11004 ; 11005 ; Disk work areas 11006 ; 11007 ; These are used by the BDOS to detect any unexpected 11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 11010 ; 23B0 11011 Disk*A\$Workarea: DS 32 ; A: 23B0 11012 Disk*B\$Workarea: DS 32 ; B: 23F0 11013 Disk*C*Workarea: DS 16 ; C: 2400 11014 Disk*D\$Workarea: DS 16 ; C: 2400 11014 Disk*D\$Workarea: DS 16 ; C: 11015 ; 11016 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1.</pre>	2330	11002	M\$Disk\$Buffer:		DS	128	;Intermediary buffer for
<pre>11004 ; 11005 ; Disk work areas 11006 ; 11007 ; These are used by the BDOS to detect any unexpected 11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 11010 ; 23B0 11011 Disk\$A\$Workarea: DS 32 ; A: 23B0 11012 Disk\$A\$Workarea: DS 32 ; B: 23F0 11013 Disk\$C\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; C: 11015 ; 11015 ; 11015 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1.</pre>	1 .						
<pre>11005 ; Disk work areas 11006 ; 11007 ; These are used by the BDOS to detect any unexpected 11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 11010 ; 23B0 11011 Disk\$A\$Workarea: DS 32 ; A: 23B0 11012 Disk\$E\$Workarea: DS 32 ; B: 23F0 11013 Disk\$E\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; D: 11015 ; 11015 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1.</pre>							· ···===·/
<pre>11006 ; 11007 ; These are used by the BDOS to detect any unexpected 11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 11010 ; 23B0 11011 Disk\$A\$HOrkarea: DS 32 ; A: 23D0 11012 Disk\$B\$Workarea: DS 32 ; B: 23F0 11013 Disk\$C\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; D: 11015 ; 11016 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1.</pre>			, Diekum	rk aross			
<pre>i1007 ; These are used by the BDOS to detect any unexpected 11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 11010 ; 23B0 11011 Disk\$A\$Workarea: DS 32 ; A: 23B0 11012 Disk\$B\$Workarea: DS 32 ; B: 23F0 11013 Disk\$C\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; D: 11015 ; 11015 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1.</pre>	1			· ~ areas			
<pre>11008 ; change of diskettes. The BDOS will automatically set 11009 ; such a changed diskette to read-only status. 11010 ; 23B0 11011 Disk\$A\$Workarea: DS 32 ; A: 23B0 11013 Disk\$C\$Workarea: DS 32 ; B: 23F0 11013 Disk\$C\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; D: 11015 ; 11016 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11018 ; 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1.</pre>							
11009;such a changed diskette to read-only status.11010;23B011011Disk\$A\$Workarea:DS23D011012Disk\$A\$Workarea:DS23F011013Disk\$C\$Workarea:DS240011014Disk\$D\$Workarea:DS11015;11016;11017:Disk <allocation td="" vectors<="">11018;11019:11020;which allocation blocks are used and which are free.11021;0ne byte is used for eight allocation blocks, hence the11022;expression of the form (allocation blocks/8)+1.</allocation>							
<pre>11010 ; 23B0 11011 Disk\$A\$Workarea: DS 32 ; A: 23D0 11012 Disk\$B\$Workarea: DS 32 ; B: 23F0 11013 Disk\$C\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; D: 11015 ; 11015 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;</pre>							
23B0 11011 Disk\$A\$Workarea: DS 32 ; A: 23D0 11012 Disk\$E\$Workarea: DS 32 ; B: 23F0 11013 Disk\$E\$Workarea: DS 16 ; C: 2400 11014 Disk\$E\$Workarea: DS 16 ; D: 11015 ;			; such a	changed	diskette	to rea	nd-only status.
23D011012Disk\$D\$Workarea:DS32; B:23F011013Disk\$C\$Workarea:DS16; C:240011014Disk\$D\$Workarea:DS16; D:11015;11015;11017;Disk allocation vectors11018;111019;These are used by the BDOS to maintain a bit map of11020;which allocation blocks are used and which are free.11021;One byte is used for eight allocation blocks, hence the11022;expression of the form (allocation blocks/8)+1.11023;			;				
23D011012Disk\$B\$Workarea:DS32; B:23F011013Disk\$C\$Workarea:DS16; C:240011014Disk\$D\$Workarea:DS16; D:11015;11016;11017;Disk allocation vectors11018;11019;11019;These are used by the BDOS to maintain a bit map of11020;which allocation blocks are used and which are free.11021;One byte is used for eight allocation blocks, hence the11022;expression of the form (allocation blocks/8)+1.11023;	2380	11011	Disk\$A\$Workarea	:	DS	32	; A:
23F0 11013 Disk\$C\$Workarea: DS 16 ; C: 2400 11014 Disk\$D\$Workarea: DS 16 ; D: 11015 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;	2300	11012	Disk\$B\$Workarea	:	DS	32	; B:
2400 11014 Disk\$D\$Workarea: DS 16 ; D: 11015 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;							
<pre>11015 ; 11016 ; 11017 ; Disk allocation vectors 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;</pre>							
<pre>11016 ; 11017 ; Disk allocation vectors 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;</pre>	1			-			,
<pre>11017 ; Disk allocation vectors 11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;</pre>	1		-				
<pre>11018 ; 11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;</pre>	1		,	1003445-			
11019 ; These are used by the BDOS to maintain a bit map of 11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;				rocation	vectors		
11020 ; which allocation blocks are used and which are free. 11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;							
<pre>11021 ; One byte is used for eight allocation blocks, hence the 11022 ; expression of the form (allocation blocks/8)+1. 11023 ;</pre>	1						
<pre>11022 ; expression of the form (allocation blocks/8)+1. 11023 ;</pre>	1						
11023 ;							
	1	11022	; express	ion of t	he form	(alloca	ition blocks/8)+1.
2410 11024 Disk\$A\$Allocation\$Vector DS (174/8)+1 ; A:		11023	;				
	2410		Disk\$A\$Allocati	on\$Vecto	r	DS	(174/8)+1 ; A:

2426	11025	Disk\$B\$Allocation\$Vector	DS	(174/8)+1	; B:
	11026	;			
243C	11027	Disk\$C\$Allocation\$Vector	DS	(242/8)+1	; C:
245B	11028	Disk\$D\$Allocation\$Vector	DS	(242/8)+1	; D:
	11029	;			
247A	11030	M\$Disk\$Allocation\$Vector	DS	(192/8)+1	; M\$Disk
	11031				
2493	11032	END : of enhanced	BIOS lis	ting	

Figure 8-10. (Continued)

Г

Classes of Errors BIOS Error-Handling Functions Practical Error Handling Character I/O Errors Disk Errors Improving Error Messages



Dealing with Hardware Errors

This chapter describes the enhancements you can make to improve CP/M's somewhat primitive error handling. It covers the general classes of errors that the BIOS may have to handle. It describes some of the underlying philosophical aspects of errors, how to detect them, and how to correct them or otherwise make the best of the situation.

At the end of the chapter are some example error-handling subroutines. Some of these have already been shown in the previous chapter as part of the enhanced BIOS (Figure 8-10); they are repeated here so that you can see them in isolation.

Classes of Errors

Basically, the user perceives only two classes of errors—those that are usercorrectable and those that are not. There is a third, almost invisible class of errors—those that are recoverable by the hardware or software without the user's intervention. The possible sources for hardware errors vary wildly from one computer system to another, since error detection is heavily dependent on the particular logic in the hardware. The BIOS can detect some hardware-related errors — mainly errors caused when something takes too long to happen, such as when a recalcitrant printer does not react in a specified length of time.

The BDOS has no built-in hardware detection code. It can detect *system* errors, such as an attempt to write to a disk file that is marked "Read-Only" in the file directory or attempts to access files that are not on the disk. These BDOS-detected errors, however, generally are unrelated to the well-being of the hardware. For example, a disk controller with a hardware problem could easily overwrite a sector of the directory, thereby deleting several files. This error would not show up until the user tried to use one of the now-departed files.

BIOS Error-Handling Functions

The error-handling code in the BIOS has to serve the following functions:

- Detection
- Analysis
- Indication
- · Correction.

Error Detection

Clearly, before any later steps can be taken, an error must be detected. This can be done by the software alone or by the BIOS interacting with error-detecting logic in the hardware. In general, the only errors that the BIOS can detect unassisted are caused when certain operations take longer to complete than expected. Because the writer of the BIOS knows the operating environment of the specific peripherals in the system, the code can predict how long a particular operation should take and can signal an error when this time is exceeded. This would include such problems as printers that fail to react within a specified time period.

The BIOS can work in cooperation with the hardware to determine whether the hardware itself has detected an error. Armed with the hardware's specifications, the BIOS can input information on controller or device status to trigger error-detecting logic. How this should be done depends heavily on the peripheral devices in your computer system and the degree to which these devices have "smart" controllers capable of processing independently of the computer. Unfortunately, many manufacturers document the significance of individual status bits that indicate errors, but not combinations of errors, or what to do when a particular error occurs.

Error Analysis

Given that your BIOS has detected an error, it must first determine the class of error; that is, whether or not the error can be corrected by simply trying the operation again. Some errors appear at first to be correctable, but retrying the operation several times still fails to complete it. An example would be a check-sum error while reading a disk sector. If several attempts to read the sector all yield an error, then it becomes a "fatal" error. The code in your BIOS must be capable of initial classification and then subsequent reclassification if remedial action fails.

Other types of errors can be classified immediately as fatal errors—nothing can be done to save the situation. For example, if the floppy disk controller indicates that it cannot find a particular sector number on a diskette (due to an error in formatting), there is nothing that the BIOS can do other than inform the user of the problem and supply other helpful information.

Analysis of errors may require some basic research, such as inducing failures in the hardware and observing combinations of error indicators. For example, some printers (interfaced via a parallel port) indicate that they are "Out of Paper" or "Busy" when, in fact, they are switched off. The BIOS should detect this condition and tell the user to switch the printer on, not load more paper.

Error Indication

An incomplete or cryptic error message is infuriating. It is the functional equivalent of saying, "There has been an error. See if you can guess what went wrong!"

An error message, to be complete, should inform the recipient of the following:

- The fact that an error has occurred.
- Whether or not automatic recovery has been attempted and failed.
- The details of the error, if need be in technical terms to assist a hardware engineer.
- What possible choices the user has now.

To put these points into focus, consider the error message that can be output by CP/M after you have attempted to load a program by entering its name into the CCP. What you see on the console is the following dialog:

```
A><u>myprog<cr></u>
BAD LOAD
A>
```

All you know is that there has been an error, and you must guess what it is, even though the specific cause of the error was known to CP/M when it output the message. This error message is output by the CCP when it attempts to load a

".COM" file larger than the current transient program area. The message "BAD LOAD" is only understandable *after* you know what the error is. Even then, it does not tell you what went wrong, whether there is anything you can do about it, and how to go about doing it.

To be complete, this error message could say something like this:

A>myprog<cr>

```
"MYPROG.COM" exceeds the available memory space by
1,024 bytes, and therefore cannot be loaded under the
current version of CP/M.
```

Notice how the message tells you what the problem is, and even quantifies it so that you can determine its severity (you need to get 1K more memory or reduce the program's size). It also tells you how you stand—you cannot load this program under the current version of CP/M, so retrying the operation is futile.

Not many systems programmers like to output messages like the example above. They argue that such a message is too long and too much work for something that does not happen often. Admittedly, the message *is* too long. It could be shortened to read

(131) Program 1,024 bytes too large to load.

This conveys the same information; the number in parentheses can serve as a reference to a manual where the full impact of the message should be described.

The major problem with the way error messages are designed is that they usually are written by programmers to be read by nontechnical lay users, and programmers are notoriously bad at guessing what nonexperts need to know.

Error indications you design should address the following issues, from the point of view of the user:

- The cause of the error
- The severity of the error
- The corrective action that has and can be taken.

Examine the error messages in the error processor for the example BIOS in Figure 8-10, from line 03600 onward. Although these are an improvement on the BDOS all-purpose

BDOS Error on A: Bad Sector

even these messages do not really meet all of the requirements of a good error message system.

Another often overlooked aspect of errors is that most hardware errors form a pattern. This pattern is normally only discernible to the trained eye of a hardware maintenance engineer. When these engineers are called to investigate a problem,

they will quiz the user to determine whether a given failure is an isolated incident or part of an ongoing pattern. This is why an error message should contain additional technical details. For example, a disk error message should include the track and sector used in the operation that resulted in an error. Only with these details can the engineer piece together the context of a failure or group of failures.

Error Correction

Given that a lucid error message has been displayed on the console, the user is still confronted with the question: "Now what do I do?" Not only can this be difficult for the user to answer, but also the particular solution decided upon can be hard for the BIOS to execute.

Normally, there are three possible options in response to errors:

- Try the operation again
- · Ignore the error and attempt to continue
- Abort the program causing the error and return to CP/M.

For some errors, retrying can be effective. For example, if you forget to put the printer on-line and get a "Printer Timeout" error message, it is easy to put the printer back on-line and ask the BIOS to try again to send data to the printer.

Seldom can you ignore an error and hope to get sensible results from the machine; many disk controllers do not even transfer data between themselves and the disk drive if an error has been detected. Only ignorant users, or brave ones in desperation, ignore errors.

Aborting the program causing the error is a drastic measure, although it does escape from what could otherwise be a "deadly embrace" situation. For example, if you misassign the printer to an inactive serial port and turn on printer echoing (with the CONTROL-P toggle), you will send the system into an endless series of "Printer Timeout" messages. If you abort the program, the error handler in the BIOS executes a System Reset function (function 0) in the BDOS, CP/M warm boots, and control is returned to the CCP. In the process, the printer toggle is reset and the circle is broken.

Practical Error Handling

This section discusses several errors, describing their causes and the way in which the BIOS and the user can handle them when they occur.

Character I/O Errors

At the BIOS level, most detectable errors related to character input or output will be found by the hardware chips.

Parity Error

Parity, in this context, refers to the number of bits set to 1 in an 8-bit character. The otherwise unused eighth bit in ASCII characters can be set to make this number always odd, or alternatively, always even. Your computer hardware can be programmed to count the number of 1 bits in each character and to generate an error if the number is odd (odd parity) or, alternatively, if it is even (even parity). If the hardware on the other end of the line is programmed to operate in the same mode, parity checking provides a primitive error-detection mechanism—you can tell that a character is bad, but not what it should have been.

CP/M does not provide a standard mechanism for reporting a parity error, so your only option is to reset the hardware and substitute an ASCII DEL (7FH; delete) character in the place of the erroneous character.

If your BIOS is operating in a highly specialized environment, you may need to count the number of such parity errors so that a utility program can report on the overall performance of the system.

Framing Error

When an 8-bit ASCII character is transmitted over a serial line, the eight bits are transmitted serially, one after the other. A *start* bit is transmitted first, followed by the data character and then a *stop* bit. If the hardware fails to find the stop and start bits in the correct positions, a *framing error* will occur. Again, the only option available to the BIOS is to reset the hardware chip and substitute an ASCII DEL.

Overrun Error

This error occurs when incoming data characters arrive faster than the program can handle them, so that the last characters overrun those being processed by the hardware chip. This error can normally be avoided by the use of serial line protocols, such as those in the example BIOS in Figure 8-10.

An overrun error implies that the protocol has broken down. As with the parity and framing errors, almost the only option is to reset the hardware and substitute a DEL character.

Printer Timeout Error

This is one of the few errors where the BIOS can sensibly attempt an error recovery. The error occurs when the BIOS tries to output a character to a serial printer and finds that the printer is not ready for more than, say, 30 seconds. The most common cause of this error is that the user forgets to put the printer on-line. Many printers require that they be off-line during a manual form feed, and users will often forget to push the on-line button afterward.

After a 30-second delay, the BIOS can send a message to the console device(s) informing the user of the error and asking the user to choose the appropriate course of action. Note that console output can be directed to more than one device.

Parallel Printers

Printers connected to your system by means of a parallel port can indicate their status to the computer much more easily than can serial printers. They can communicate such error states as "Out of Paper," "End of Ribbon," and "Off-line."

These single-error indicators can also be used in combination to indicate whether the printer cable is connected, or even whether the printer is receiving power. You need to experiment, deliberately putting the printer into these states and reading status in order to identify them. It is misleading to indicate to the inexperienced user that the printer is "Out of Paper" when the problem is that the data cable has inadvertently become disconnected.

However, each of these errors can be dealt with in the same way as the serial printer's timeout problem: display an error message and request the user's choice of action.

Example Printer Error Routine

Figure 9-1 shows an example of a program that handles printer errors. It consists of several subroutines, including

- · The error detection classification and indication routine
- The error correction routine.

It uses other subroutines that are omitted from the figure to avoid obscuring the logic. These subroutines are listed in full in the example BIOS in Figure 8-10.

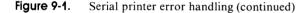
This example shows, in outline form, how to handle the situation when a serial printer remains busy for too long. It is intended that this generic example show how to deal with this class of errors. 2 ; . ; 7 The example presupposes the existence of a clock interrupt ; every 16.666 milliseconds (1/60th of a second), and that control will be transferred to the Real Time Clock service routine each time the clock "ticks". Figure 8-10 shows a more complete example, installed in a real BIOS. ;BDOS system reset function
;BDOS entry point 0000 = B\$System\$Reset FOIL 0 0005 = BDOS EQU 5 0000 00 Printer\$Timeout\$Flag: DB 0 ;This flag is set by the interrupt ; service subroutine that is called when the watchdog timer subroutine count hits zero (after having counted down a 30-second delay) 0708 = ;Given a clock period of 16.666 ms Printer\$Delay\$Count EQU 1800 this represents a delay of 30 secs ;

Figure 9-1. Serial printer error handling

000D 000A	=	CR LF			EQU EQU	odh Oah	;Carriage return ;Line feed
		; Printer:	\$Busy\$Me	sage:			
0003	0D0A 5072696E74 436865636B	Ļ		CR,LF 'Printer	r has be that it	en busy is on-li	for too long,´,CR,LF ne and ready.´,CR,LF,O
004E		; Printer: ;	\$Characte	?r:	DB	0	;Save area for the data character ; to be output
		; LIST:	;				;<=== Main BIOS entry point ;<=== I/O redirection code occurs here
004F 0050	79 324E00		MOV STA	A,C Printer	\$Charact	ler	;Save the data character
0053	010807	Printer	\$Retry: LXI	B,Printe	er\$Delay	/\$Count	;This is the count of the number ; of clock ticks before the watchdog ; subroutine call
	217E00 CDA300		LXI CALL	H,Printo Set\$Wate		J\$Out	; <== this address ;Sets the watchdog running
005C	CDA300	Printer	\$Wait: CALL	Get\$Pri	nter\$Sta	atus	;See if the printer is ready to ; accept a character for output ; This includes checking if the printer ; is "Busy" because the driver is ; waiting for XON, ACK, or DTR to
005F	C26C00		JNZ	Printer	\$Ready		; come high ;The printer is now ready
	3A0000		LDA	Printer	₿Timeout	:\$Flag	;Check if the watchdog timer has ; hit zero (if it does, the ; watchdog routine will call ; the Printer\$Timed\$Out code ; that sets this flag)
0065 0066	B7 C28400		ORA JNZ	A Display	\$Busy\$Me	ssage	;Yes, so display message to ; indicate an error has occurred
0069	C35C00		JMP	Printer	≸Wait		Otherwise, check if printer is
		Printer	\$Ready:				;The printer is now ready to output ; a character, but before doing so, ; the watchdog timer must be reset
006C 006D 0070 0073	010000 CDA300		DI LXI CALL EI	B,0 Set\$Wat	chdog		;Ensure no false timeout occurs ;This is done by setting the count ; to zero
0077	3A4E00 11A300 CDA300		LDA LXI CALL	Printer D,Printo Output\$	er\$Devic	e\$Table	;Get character to output ;DE -> device table for printer ;Output the character to the printer
007D			RET				Return to the BIOS's caller;
007E		;	\$Timed\$O	A, OFFH			;Control arrives here from the ; watchdog routine if the ; watchdog count ever hits zero ; This is an interrupt service ; routine ;All registers have been saved ; before control arrives here ;Set printer timeout flag
0080 0083	320000 C9		STA RET	Printer	\$Timeout	\$Flag	Return back to the watchdog;
							:Interrupt service routine

Figure 9-1. (Continued)

```
,
Display$Busy$Message:
                                                         Printer has been busy for
                                                            30 seconds or more
0084 AF
                        XRA
                                                         Reset timeout flag
0085 320000
                        STA
                                Printer$Timeout$Flag
                                H,Printer$Busy$Message
0088 210100
                        IXT
                                                         ;Output error message
008B CDA300
                       CALL
                                Output$Error$Message
008E CDA300
                        CALL
                                Request$User$Choice
                                                         ;Displays a Retry, Abort, Ignore?
                                                         ; prompt, accepts a character from
                                                            the keyboard, and returns with the
                                                            character, converted to upper
                                                            case in the A register
0091 FE52
                       CPI
                                'R'
                                                         Check if Retry
0093 CA5300
                        JZ
                                Printer$Retry
0096 FE41
                        CPI
                                ' A '
                                                         :Check if Abort
0098 CA9E00
                        JZ
                                Printer$Abort
009B FE49
009D C8
                       CP I
RZ
                                111
                                                         ;Check if Ignore
               Printer$Abort:
009E 0E00
                                C,B$System$Reset
                       MVI
                                                         :Issue system reset
00A0 C30500
                                BDOS
                       . IMP
                                                         ;No need to give call as
                                                         ; control will not be returned
                       Dummy subroutines
                        These are shown in full in Figure 8-10. The line numbers in
                       Figure 8-10 are shown in the comment field below
               Printer$Device$Table:
                                                 :Line 01300 (example layout)
               Request$User$Choice:
                                                 :Line 03400
               Output$Error$Message:
                                                 :Line 03500
               Get$Printer$Status:
                                                 ;Line 03900 (similar code)
               Output$Data$Byte:
                                                 ;Line 05400 (similar code)
               Set$Watchdog:
                                                 :Line 05800
```



Disk Errors

Disks are much more complicated than character I/O devices. Errors are possible in the electronics and in the disk medium itself. Most of the errors concerned with electronics need only be reported in enough detail to give a maintenance engineer information about the problem. This kind of error is rarely correctable by retrying the operation. In contrast, media errors often can be remedied by retrying the operation or by special error processing software built into the BIOS. This chapter discusses this class of errors.

Media errors occur when the BIOS tries to read a sector from the disk and the hardware detects a check-sum failure in the data. This is known as a *cyclical redundancy check* (CRC) error. Some disk controllers execute a read-after-write check, so a CRC error can also occur during an attempt to write a sector to the disk.

With floppy diskettes, the disk driver should retry the operation at least ten times before reporting the error to the user. Then, because diskettes are inexpensive and replaceable, the user can choose to discard the diskette and continue with a new one.

With hard disks, the media cannot be exchanged. The only way of dealing with bad sectors is to replace them logically, substituting other sectors in their place.

There are two fundamentally different ways of doing this. Figure 9-2 shows the scheme known as sector sparing—substituting sectors on an outer track for a sector that is bad.

The advantage of this scheme is that it is dynamic. If a sector is found to be bad in a read-after-write check, even after several retries, then the data intended for the failing sector can be written to a spare sector. The failing sector's number is placed into a spare-sector directory on the disk. Thereafter, the disk drivers will be redirected to the spare sector every time an attempt is made to read or write the bad sector.

The disadvantage of this system is that the read/write heads on the disk must move out to the spare sector and then back to access the next sector. This can be a problem if you attempt to make a high-speed backup on a streaming tape drive (one that writes data to a tape in a single stream rather than in discrete blocks). The delay caused by reading the spare sector interrupts the data flow to the streaming tape drive.

You need a special utility program to manipulate the spare-sector directory, both to substitute for a failing sector manually and to attempt to rewrite a spare sector back onto the bad sector.

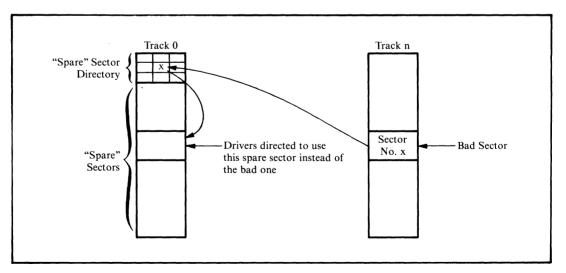


Figure 9-2. Sector sparing

Figure 9-3 shows another scheme for dealing with bad sectors. In this method, bad sectors are skipped rather than having sectors substituted for them.

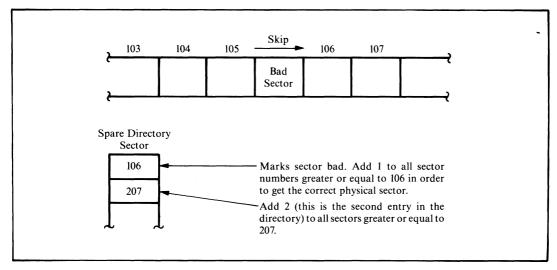
The advantage of sector skipping is that the heads do not have to perform any long seeks. The failing sector is skipped, and the next sector is used in its place. Because of this, sector skipping can give much better performance. Data can be read off the disk fast enough to keep a streaming tape drive "fed" with data.

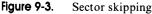
The disadvantage of sector skipping is that it does not lend itself to dynamic operation. The bad sector table is best built during formatting. Once data has been written to the disk, if a sector goes bad, all subsequent sectors on the disk must be "moved down one" to make space to skip the bad sector. On a large hard disk, this could take several minutes.

Example Bad Sector Management

Sector sparing and sector skipping use similar logic. Both require a sparesector directory on each physical disk, containing the sector numbers of the bad sectors. This directory is read into memory during cold start initialization. Thereafter, all disk read and write operations refer to the memory-resident table to see if they are about to access a bad sector.

For sector sparing, if the sector about to be read or written is found in the spare directory, its position in the directory determines which spare sector should be read.





In the case of sector skipping, every access to the disk makes the driver check the bad sector directory. The directory is used to tell how many bad sectors exist between the start of the disk and the failing bad sector. This number must be added to the requested track and sector to compensate for all the bad sectors.

The physical low-level drivers need four entry points:

- Read the specified sector without using bad sector management. This is used to read in the spare directory itself.
- Write the specified sector without using bad sector management. This is used to write the spare directory onto the disk, both to initialize it and to update it.
- Read and write the sector using bad sector management. These entry points are used for normal disk input/output.

Figure 9-4 shows the code necessary for both sector sparing and (using conditional code) sector skipping.

This example shows the modifications to be made in order to implement bad sector management using sector sparing : ; and sector skipping. : 2 0000 = False EQU Not False FFFF = True EQU . Sector\$Sparing EQU False 0000 =FFFF = Sector\$Skipping EQU Not Sector\$Sparing Additional equates and definitions Spare\$Directories: ;Table of spare directory addresses ;Note: The directories themselves ; are declared at the end of the ; BIOS 0000 D500 DW Spare\$Directory\$0 Physical disk O 0002 9701 nμ Spare\$Directory\$1 ;Physical disk 1 Spare\$Dir\$In\$Memory: ;Flags used to indicate whether spare 0004 00 ; directory for a given physical disk ; has been loaded into memory. Set by SELDSK DB Ô 0005 00 ō **DB** . 0000 = Spare\$Track EQU 0 ;Track containing spare directory sectors 0004 = Spare\$Sector FOU 4 Sector containing directory First\$Spare\$Sector Spare\$Sector + 1 0005 =FOU . Variables set by SELDSK Selected\$Spare\$Directory: 0006 0000 υW 0 ;Pointer to directory 0008 00 DB 0 ;Logical disk number Selected\$Disk: 0009 00 DB ;Floppy/hard disks Disk\$Type: 0 ;Deblocking flag 000 A 000 Deblocking\$Required: **DB** 0 0 Physical disk number 000B 00 Selected\$Physical\$Disk: DB 0 ;) These variables are part of the command 0000 0000 Disk\$Track: DW ;) These variables are pair of the controller ;) block handed over to the disk controller ō 000E 00 Disk#Sector: DB

Figure 9-4. Bad sector management

```
8000 =
                 Maximum$Track
                                            EQU
                                                    32768 ;Used as a terminator
0012 =
                 Sectors$Per$Track
                                            EQU
                                                    18
0000 =
                 First$Sector$On$Track
                                            FQU
                                                    0
                 .
                 .
Disk$Parameter$Headers:
                          .
                          ;Standard DPH Declarations
                          .
                 ;
                 ;
                          Equates for disk parameter block
                 ;
                 ;
                 ;
                         The special disk parameter byte that precedes each disk
                         parameter block, needs to be rearranged so that a
                 ;
                         physical disk drive number can be added.
                 ;
                          Disk types
                                           vvvv--- Physical disk number
0$001$0000B ;5 1/4" mini floppy
0$010$0000B ;8" floppy (SS SD)
0010 =
                 Floppy$5
                                  EQU
0020 =
                 Floppy$8
                                  EQU
0030 =
                 M$Disk
                                  EQU
                                            0$011$0000B
                                                             ;Memory disk
0040 =
                 H$Disk$10
                                  EQU
                                           0$100$0000B
                                                             ;Hard disk - 10 megabyte
0070 =
                                           FOU
                 Disk$Type$$Mask
                                                    0$111$0000B
                                                                     ;Masks to isolate values
000F =
                 Physical$Disk$Mask
                                           EQU
                                                    0$000$1111B
                          Blocking/deblocking indicator
0080 =
                 Need$Deblocking EQU
                                           1$000$0000B
                                                          Sector size > 128 bytes
                 :
                         Disk parameter blocks
                 ;
                 ;
                          . ______
                          ; Standard DPB's for A: and B:
                          :-----
                 ;
                                                    ;Logical disk C:
                                                    ;Extra byte indicates disk type
                                                    ; deblocking requirements and physical
                                                    ; disk drive.
000F C0
                         DB
                                  H$Disk$10 + Need$Deblocking + 0 ; Physical drive 0
                 Hard$5$Parameter$Block$C:
                          ;Standard format parameter block
                          :---
                 ;
                 ;
0010 CO
                         DB
                                 H$Disk$10 + Need$Deblocking + 0 ; Physical drive 0
                Hard$5$Parameter$Block$D:
                         ....
                         Standard format parameter block
                          .
                .
                Number$of$Logical$Disks
0004 =
                                                  FOU
                                                            4
                SELDSK:
                                           ;Select disk in register C
                                           ; C = 0 for drive A, 1 for B, etc.
;Return the address of the appropriate
; disk parameter header in HL, or 0000H
                                           ; if the selected disk does not exist.
0011 210000
                         LXI
                                  н, о
                                                    ;Assume an error
;Check if requested disk valid
0014 79
                         MOV
                                  A.C
0015 FE04
0017 D0
                         CPI
                                  Number$of$Logical$Disks
                                                    ;Return if > maximum number of disks
                         RNC
```

Figure 9-4. (Continued)

0018	320800	STA	Selected\$Disk	;Save selected disk number ;Set up to return DPH address
001B	6F	MOV	L,A	;Set up to return DPA address ;Make disk into word value
0010	2600	MVI	Н, О	
				;Compute offset down disk parameter
				; header table by multiplying by
0015	20	DAD	ы	; parameter header length (16 bytes)
001E 001F	∠7 29	DAD DAD	н	;*2 ;*4
0020	29	DAD	н	;*8
0021		DAD	н	;*16
0022	110F00	LXI	D,Disk\$Paramete	
0025		DAD	D	;DE -> appropriate DPH
0026	E5	PUSH	н	;Save DPH address
				Access disk parameter block in order;
				; to extract special prefix byte that
				; identifies disk type and whether
				; deblocking is required
0007	110400		B 10	; .Cat DPD painter offsat in DDU
0027 002A	110A00	LXI DAD	D,10 D	;Get DPB pointer offset in DPH :DE -> DPB address in DPH
002A 002B		MOV	U E, M	;DE -> DPB address in DPH ;Get DPB address in DE
0020		INX	н.	yeet and address in an
002D		MOV	D, M	
002E		XCHG		;DE -> DPB
		SELDSK\$Set\$Dis	k\$Tvoe∙	
002F	2B	DCX	H H	;DE -> prefix byte
0030		MOV	A,M	;Get prefix byte
0031		ANI	Disk\$Type\$Mask	
0033	320900	STA	Disk\$Type	;Save for use in low-level driver
0036		MOV	Α,Μ	;Get another copy of prefix byte
0037		ANI	Need\$Deblocking	
0039	320A00	STA	Deblocking\$Requ	ired ;Save for use in low-level driver
				;Additional code to check if spare ; directory for given disk has already ; been read in.
0000	70	MOL	A M	·Cot shusias) disk sumbar
003C 003D		MOV ANI	A,M Physical\$Disk\$M	;Get physical disk number Mask
	320B00	STA		ask al\$Disk ;Save for low-level drivers
0042		MOV	E,A	;Make into word
0043		MVI	D,O	Manager
0045 0048	210400 19	LXI DAD	H,Spare\$Dir\$In\$ D	Memory ;Make pointer into table
0040	• *	500	2	
0049		MOV	A, M	;Get flag
004A		ORA	A	·Convo divertory already in memory
004B 004E	C27700	JNZ INR	Dir\$In\$Memory M	;Spare directory already in memory ;Set flag
~~+C		TIAL	.,	,
	210000	LXI	H,Spare\$Directo	
0052		DAD	D	; spare directory (added twice
0053	19	DAD	D	; as table has word entries)
				;HL -> word containing directory addr.
0054		MOV	E,M	
0055		INX	Н	;Spare directory address in DE
0056 0057		MOV XCHG	D, M	;Spare directory address in DE ;HL -> spare directory
0007		X0110		,
0058	220600	SHLD	Selected\$Spare\$	Directory ;Save for use in physical ; drivers later on
	110000	LXI	D,Spare\$Track	;Track containing spare directory
	3A0B00	LDA	Selected\$Physic	cal\$Disk
0061		MOV	B,A	_
0062		MVI	A, Spare\$Sector	;Sector containing spare directory
0064		MVI CALL	C,Spare\$Length/ Absolute\$Read	/8 ;Number of bytes in spare directory / 8 ;Read in spare directory - without
0044		UMLL	nu sorare svedu	
0066	00000			; using bad sector management

Figure 9-4. (Continued)

```
0069 2A0600
                        LHLD
                                 Selected$Spare$Directory ;Set end marker
006C 11C000
006F 19
                         LXI
                                 D,Spare$Length
                                                          ; at back end of spare directory
                         DAD
                                 n
                                                           ;Use maximum track number
0070 110080
                        IXT
                                 D.Maximum$Track
0073 73
                        MOV
                                 M,E
0074 23
                         TNX
                                 H
0075 3602
                                 M, D
                        MVI
                Dir$In$Memory:
0077 E1
                        POP
                                 н
                                                  ;Recover DPH pointer
0078 C9
                        RET
                ;
                ;
                        In the low-level disk drivers, the following code must be
                ;
                        inserted just before the disk controller is activated to
                ;
                ;
                        execute a read or a write command.
                ;
0079 240000
                                 Disk$Track
                                                           ;Get track number from disk
                        LHLD
                                                           ; controller command table
                                                           ;DE = track
007C EB
                         XCHG
                                 Selected$Spare$Directory ;HL -> spare directory
007D 2A0600
                         LHLD
0080 2B
                         DCX
                                 н
                                                           ;Back up one entry
0081 2B
                         DCX
                                 н
                                                           ; (3 bytes)
0082 2B
                         DCX
                                 н
0083 3A0E00
                         LDA
                                 Disk$Sector
                                                           ;Get sector number
                                                           ;Save for later
0086 4F
                        MOV
                                 C,A
                                                           ;Set counter (biased -1)
0087 06FF
                        MVI
                                 B. OFFH
                Check$Next$Entry:
                                                           ;Update to next (or first) entry
0089 23
                         INX
                                 н
                Check$Next$Entry1:
008A 23
                        INX
                                 н
                Check$Next$Entry2:
008B 23
                         INX
                                 'н
                                                           :Update count
008C 04
                         INR
                                 B
                         1F
                                 Sector$Sparing
                                                           :If sparing is used, the
                                                           ; end of the table is indicated
                                                           ; by an entry with the track number
                                                              = to maximum track number
                         LXI
                                 D, Maximum$Track
                                                           ;Get maximum track number
                         CALL
JZ
                                  CMPM
                                                            ;Compare DE to (HL),
                                                                                  (HL+1)
                                 Not$Bad$Sector
                                                           ;End of table reached
                         ENDIF
                                                           ;Note: For sector skipping
                                                           ; the following search loop will
; terminate when the requested track
                                                           ; is less than that in the table.
;This will always happen when the
                                                            ; maximum track number is encountered
                                                            ; at the end of the table.
008D EB
                         XCHG
                                                           ;DE -> table entry
008E 2A0C00
                         LHLD
                                 Disk$Track
                                                           ;Get requested track
0091 EB
                         XCHG
                                                           ;DE = req. track, HL -> table entry
0092 CDCD00
                         CALL
                                 CMPM
                                                           ;Compare req. track to table entry
                        1F
                                 Sector$Sparing
                                                           ;Use the following code for
                                                            ; sector sparing
                         . IN7
                                                           ;Track does not match
;HL -> MS byte of track
                                 Check$Next$Entry
                         INX
                                 н
                         INX
                                 H
                                                           ;HL -> sector
                         MOV
                                 A,C
                                                           ;Get requested sector
                         CMP
                                 м
                                                           ;Compare to table entry
                         JNZ
                                 Check$Next$Entry2
                                                           Sector does not match
                                                           ;Track and sector match, so
                                                              substitute spare track and
                                                           ;
                                                           ; appropriate sector
```

Figure 9-4. (Continued)

;Get track number used for spare LXI H, Spare\$Track sectors ;Substitute track SHLD Disk\$Track :Get first sector number MVI A, First\$Spare\$Sector ;Add on matched directory ADD R ; entry number ;Substitute sector STA Disk\$Sector ENDIF Sector\$Skipping ;Use the following code for IE sector skipping ; The object is to find the ; entry in the table which ; is greater or equal to the ; requested sector/track ;Possible match of track and sector ;Requested track ;Requested track > table entry JZ Tracks\$Match 0095 CA9E00 JNC Compute\$Increment 0098 D2AC00 Check\$Next\$Entry 009B C38900 JMF Tracks\$Match: ;HL -> MS byte of track ;HL -> sector ;Get sector from table 009E 23 INX н 009F 23 00A0 77 INX н MOV M.A 00A1 B9 CMP ;Compare with requested sector С 00A2 CAABOO 00A5 D2AC00 00A8 C38B00 JZ Sectors\$Match ;Track/sector matches - INC Compute\$Increment ;Req. trk/sec < spare trk/sec ;Move to next table entry Check\$Next\$Entry2 . IMP Sectors\$Match: 00AB 04 INR в ;If track and sectors match with ; a table entry, then an additional ; sector must be skipped Compute\$Increment: ;B contains number of cumulative ; number of sectors to skip ;Get requested sector 00AC 79 MOV A,C 00AD 80 ADD R ;Skip required number ;Determine final sector number ; and track increment 00AE 0612 MVI B. Sectors \$Per \$Track DIV\$A\$BY\$B ;Returns C = quotient, A = remainder OOBO CDC300 CALL 00B3 320E00 STA Disk\$Sector ;A = new sector number 00B6 59 00B7 1600 MOV E.C ;Make track increment a word MUT **D**. O Disk\$Track 00B9 2A0C00 00BC 19 LHLD ;Get requested track DAD :Add on increment n 00BD 220C00 SHLD Disk\$Track Save updated track ENDIF Not\$Bad\$Sector: ;Either track/sector were not bad, ; or requested track and sector have
; been updated. 00C0 C3D500 JMP Read\$Write\$Disk ;Go to physical disk read/write ; IF Sector\$Skipping ;Subroutine required for skipping : routine ; ; DIV\$A\$BY\$B : Divide A by B ; ş This routine divides A by B, returning the quotient in C ; and the remainder in A. ; Entry parameters ; ; A = dividend 2 B = divisor ; ; : Exit parameters

Figure 9-4. (Continued)

```
Ŧ
                   ;
                                       A = remainder
                                       C = quotient
                   :
                   DIV$A$BY$B:
                                     'c,o
00C3 0E00
                             MVI
                                                           ;Initialize quotient
                   DIV$A$BY$B$Loop:
0005 00
                             INR
                                        с
                                                           ;Increment quotient
00C6 90
                             SUB
                                       в
                                                            ;Subtract divisor
00C7 F2C500
                             . IP
                                       DIV$A$BY$B$Loop ;Repeat if result still +ve
OOCA OD
                             DCR
                                                            Correct quotient
                                       С
00CB 80
00CC C9
                             ADD
                                       R
                                                            ;Correct remainder
                             RET
                             ENDIF
                   :
                             CMPM
                   ;
                             Compare memory
                   :
                   ;
                             This subroutine compares the contents of DE to (HL) and (HL+1)
                   ;
                             returning with the flags as though the subtraction (HL) - DE
                   ;
                             were performed.
                   ;
                             Entry parameters
                   :
                   :
                                       HL -> word in memory
DE = value to be compared
                   :
                   ;
                   :
                             Exit parameters
                   ;
                   :
                                       Flags set for (HL) - DE
                   ;
                   CMPM:
OOCD 7E
                             MOV
                                       A,M
                                                                      ;Get MS byte
OOCE BA
                             CMP
                                       D
OOCF CO
                             RN7
                                                                      ;Return now if MS bytes unequal
;HL -> LS byte
;Get LS byte
00D0 23
00D1 7E
                             TNX
                                       н
                                       A,M
                             MOV
00D2 BB
                             CMP
                                       E
00D3 2B
                             DCX
                                       н
                                                                      ;Return with HL unchanged
00D4 C9
                             RET
                   .
                   Absolute$Read:
                                                 ;The absolute read (and write) routines
                                                 ; access the specified sector and track
; without using bad sector management.
                   ;
                             Entry parameters
                   :
                                       HL -> Buffer
DE = Track
                   ;
                   :
                                       A = Sector
                   ;
                                       B = Physical disk drive number
                   1
                                       C = Number of bytes to read / 8
                   .
                   1
                             Set up disk controller command block with parameters in
                   .
                             registers, then initiate read operation by falling through
                             into Read$Write$Disk code below.
                  Read$Write$Disk:
                             . ----
                             The remainder of the low level disk drivers follow,
                             ; reading the required sector and track.
                             :
                  ;
                             Spare directory declarations
                   :
                   ;
                             Note: The disk format utility creates an initial spare
directory with track/sector entries for those track/sectors
that it finds are bad. It fills the remainder of the
directory with OFFH's (these serve to terminate the
searching of the directory).
                   ;
                   ;
                   ;
                   .
```

Figure 9-4. (Continued)

0000 =	; ; Spare\$Length	EQU 64 * 3	;64 Entries, 3 bytes each ; Byte 0,1 = track ; Byte 2 = sector
	Spare\$Director	y\$0:	
0005	DS	Spare\$Length	Spare directory itself
0195	DS	2	;Set to maximum track number by SELDSK as ; a safety precaution. The FORMAT utility ; puts the maximum track number into all ; unused entries in the spare directory.
	Spare\$Director	y\$1:	
0197	DS	Spare\$Length	;Spare directory itself
	DS	2	:End marker

Figure 9-4. Bad sector management (continued)

Improving Error Messages

The final extension to BIOS error handling discussed here is in disk-driver error-message handling. The subroutine shown in the example BIOS in Figure 8-10, although a significant improvement on the messages normally output by the BDOS, did not advise the user of the most suitable course of action for each error. Figure 9-5 shows an improved version of the error message processor.

```
This shows slightly more user-friendly error processor
                           for disk errors than that shown in the enhanced BIOS in Figure 8-10.
                 :
                           This version outputs a recommended course of action depending on the nature of the error detected. Code that remains unchanged from Figure 8-10 has been
                 ;
                 :
                 1
                 .
                           abbreviated.
                 .
                           Dummy equates and data declarations needed to get
                 .
                           an error free assembly of this example.
0001 =
                                              EQU
                                                        01H
                                                                 ;Read command for controller
                 Floppy$Read$Code
0002 =
                 Floppy$Write$Code
                                              EQU
                                                        02H
                                                                 ;Write command for controller
0000 00
                 Disk$Hung$Flag:
                                              DB
                                                        0
                                                                 ;Set NZ when watchdog timer times
                                                                     out
0258 =
                 Disk$Timer
                                              EQU
                                                        600
                                                                 ;10-second delay (16.66ms tick)
                                                        43H
0043 =
                 Disk$Status$Block
                                              EQU
                                                                 ;Address in memory where controller
                                                                     returns status
                                                                 ;Values from controller command table
                                              DB
                                                        0
0001 00
                 Floppy$Command:
                 Floppy$Head:
                                                        0
0002 00
                                              DB
                                              DB
0003 00
                 Floppy$Track:
                                                        0
0004 00
                 Floppy$Sector:
                                              DB
                                                        0
```

Figure 9-5. User-friendly disk-error processor

0000	00.	Deplock	ing\$Requ	ired:	DB	0	;Flag set by SELDSK according ; to selected disk type
0006	00	Disk\$Ern	ror\$Flag	:	DB	0	;Error flag returned to BDOS
0007	00	, In\$Buffo	er\$Disk:		DB	0	;Logical disk Id. relating to curren ; disk sector in deblocking buffer
		;	Equates	for Mess	ages		
0007	=	; BELL	EQU	07H	;Sound t	erminal	bell
000D 000A		CR LF	EQU EQU	ODH	;Carria ;Line fe	e retur	
0005	=	; BDOS	EQU	5	;BDOS er	ntry poi	nt (for system reset)
		; ;					
		;					
			ock\$Retr	y :			
			; and in	ed code t nitiate t	he disk	operati	ontroller command table on
0008	C31500		;	Wait\$For			
_	÷					. –	
		; Write\$PI	nysical:				contents of disk buffer to
000B	3E02		MVI	A,Floppy	\$Write\$C		ect sector ;Get write function code
0000	C31200	Read\$Ph;	JMP ysical:	Common\$P	hysical	;Read p	common code reviously selected sector
0010	3E01		MVI	A, Floppy	*Paad#Ca	; into	disk buffer ;Get read function code
0010		Common \$6	Physical		*Neau≠cc	JUE	; det read function code
AA4 A							
0012	320100		STA	Floppy\$C	ommand	;Set co	mmand table
0012	320100	; Deblock	Betry:			;Re-ent	mmand table ry point to retry after error
0012	320100		#Retry: ; ; Omitto ; and in	ed code s	ets up c the disk	;Re-ent lisk con operat	ry point to retry after error troller command block
0012	320100	Deblock:	Retry: ; ; Omitto ; and in ;	ed code s	ets up c the disk	;Re-ent disk con < operat ;Wait u ; oper ; if a	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred
0015		Deblock:	Retry: ; ; Omitto ; and in ;	ed code s nitiates	ets up c the disk	;Re-ent disk con < operat ;Wait u ; oper ; if a	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check
0015 0016	AF 320000	Deblock:	#Retry: ; Omitt; ; and in ; *\$Disk\$Co XRA STA	ed code s nitiates omplete: A Disk\$Hun	ets up c the disk g\$Flag	;Re-ent disk con coperat ;Wait u ; oper ; if a ;On ent	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear
0015 0016 0019 0010	AF 320000 213100 015802	Deblock:	<pre>#Retry: ; Omitt; ; and in ; r*Disk*Co XRA STA LXI LXI LXI</pre>	ed code s nitiates omplete: A Disk\$Hun H,Disk\$T B,Disk\$T	ets up c the disk g\$Flag imed\$Out imer	;Re-ent disk con coperat ;Wait u ; oper ; if a ;On ent	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte
0015 0016 0019 0010	AF 320000 213100	Deblock: ; Wait\$Fon	\$Retry: ; Omitic ; and i; ; and i; ; and i ; and i ; ; and i ; ; and i ; ; and i ; ; and i ; ; and i ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	ed code s nitiates omplete: A Disk\$Hun H,Disk\$T	ets up c the disk g\$Flag imed\$Out imer	;Re-ent disk con coperat ;Wait u ; oper ; if a ;On ent	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear ;Set up watchdog timer
0015 0016 0019 001C 001F 0022	AF 320000 213100 015802 CD3B03 7E	Deblock:	#Retry: ; Omitte; ; and in; *\$Disk\$Co XRA STA LXI LXI CALL it\$Loop: MOV	ed code s nitiates omplete: A Disk\$Hun H,Disk\$T Set\$Watc A,M	ets up c the disk g\$Flag imed\$Out imer	;Re-ent disk con coperat ;Wait u ; oper ; if a ;On ent	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear ;Set up watchdog timer
0015 0016 0019 001C 001F 0022 0023	AF 320000 213100 015802 CD3B03 7E	Deblock: ; Wait\$Fon	*Retry: ; omitt; and in ; *Disk*Co XRA STA LXI LXI CALL t*LCOP:	ed code s nitiates omplete: Disk\$Hun H,Disk\$T B,Disk\$T Set\$Watc	ets up c the disk g\$Flag imed\$Out imer hdog	;Re-ent disk con coperat ;Wait u ; oper ; if a ;On ent	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear ;Set up watchdog timer ;Time delay
0015 0016 0019 001C 001F 0022 0023 0024 0027	AF 320000 213100 015802 CD3803 7E B7 CA3700 3A0000	Deblock: ; Wait\$Fon	*Retry: ; ; Omitu ; and in ; *Disk*Co XRA STA LXI LXI LXI LXI CALL ti\$Loop: MOV ORA JZ LDA	ed code s nitiates omplete: Disk\$Hun H,Disk\$T B,Disk\$T Set\$Watc A,M A	ets up c the disk g\$Flag imed\$Out imer hdcg plete	;Re-ent disk con coperat ;Wait u ; oper ; if a ;On ent	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear ;Set up watchdog timer ;Time delay ;Get control byte
0015 0016 0019 001C 0022 0023 0024 0027	AF 320000 213100 015802 CD3803 7E B7 CA3700 3A0000	Deblock: ; Wait\$Fon	*Retry: ; Omitt; ; Omitt; ; and in; ;*Disk*Co XRA STA LXI LXI LXI LXI LXI LXI LXI LXI LXI LXI	ed code s nitiates Disk\$Hun H,Disk\$T B,Disk\$T Set\$Watc A,M A Disk\$Com	ets up c the disk g\$Flag imed\$Out imer hdog plete g\$Flag	;Re-ent disk con coperat ;Wait u ; oper ; if a ;On ent	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear ;Set up watchdog timer ;Time delay ;Get control byte ;Operation done
0015 0016 0017 0022 0023 0024 0027 002A 0028	AF 320000 213100 015802 CD3B03 7E B7 CA3700 3A0000 B7 C29F02	Deblock: ; Wait\$Fon	*Retry: ; Omitit; ; and ii; ; and ii; ; *Disk*Co XRA STA LXI LXI LXI LXI LXI LXI LXI LXI LA ORA JZ LDA ORA JNZ	ed code s nitiates omplete: A Disk\$Hun H,Disk\$T B,Disk\$T Set\$Watc A,M A Disk\$Com Disk\$Hun A Disk\$Hun	ets up o the disk g\$Flag imed\$Out imer hdog plete g\$Flag or	;Re-ent disk con coperat ;Wait u ; oper ; if a ;On ent	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear :Set up watchdog timer ;Time delay ;Get control byte ;Operation done ;Also check if timed out
0015 0016 0017 0022 0023 0024 0027 002A 0028	AF 320000 213100 015802 CD3B03 7E B7 CA3700 3A0000 B7	Deblock: ; Wait\$Fon	*Retry: ; ; Disk*Co XRA STA LXI LXI LXI LXI CALL it*Loop: MOV ORA JZ LDA ORA JNZ JMP	ed code s nitiates omplete: A Disk\$Hun H,Disk\$T Set\$Watc A,M A Disk\$Com Disk\$Hun A	ets up o the disk imed\$Out imer hdog plete g\$Flag or t\$Loop	;Re-ent iisk con coperat ;Wait u ; off a ;On ent ;Contro	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear :Set up watchdog timer ;Time delay ;Get control byte ;Operation done ;Also check if timed out ;Will be set to 40H
0015 0016 0012 0022 0023 0024 0027 0028 0028 0028	AF 320000 213100 015802 CD3803 7E B7 CA3700 3A0000 B7 C29F02 C32200	Deblock: ; Wait\$Fon Disk\$Wa;	*Retry: ;	ed code s nitiates A Disk\$Hun H,Disk\$T B,Disk\$T Set\$Watc A,M A Disk\$Com Disk\$Hun A Disk\$Err Disk\$Wai	ets up o the disk imed\$Out imer hdog plete g\$Flag or t\$Loop	;Re-ent isk con coperat ;Wait u ; oper ; if a ;On ent ; contro ; rout	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear ;Set up watchdog timer ;Time delay ;Get control byte ;Operation done ;Also check if timed out ;Will be set to 40H L arrives here from watchdog ine itself so this is effectively of the interrupt service routine
0015 0016 0017 0017 0022 0023 0024 0027 0024 0027 0022 0022 0022 0022	AF 320000 213100 015802 CD3803 7E B7 CA3700 3A0000 B7 C29F02 C32200 3E40	Deblock: ; Wait\$Fon Disk\$Wa;	*Retry: ; Omitti ; Omitti ; and iii ; *Disk*Cd XRA STA LXI CALL CALL CALL CALL CALL CALL ORA JZ LDA ORA JZ JMP med\$Out:	ed code s nitiates omplete: A Disk\$Hun H,Disk\$HT B,Disk\$Hun A Disk\$Com Disk\$Hun A Disk\$Hun A Disk\$Hun A	ets up c the disk g\$Flag imed\$Out imer hdcg g\$Flag or t\$Loop	;Re-ent isk con coperat ;Wait u ; oper ; if a ;On ent ; contro ; rout	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear ;Set up watchdog timer ;Time delay ;Get control byte ;Operation done ;Also check if timed out ;Will be set to 40H I arrives here from watchdog ine itself so this is effectively of the interrupt service routine ;Set disk hung error code
0015 0016 0017 0017 0022 0023 0024 0027 0024 0027 0022 0022 0022 0022	AF 320000 213100 015802 CD3B03 7E B7 CA3700 3A0000 B7 C29F02 C32200 3E40 320000	Deblock: ; Wait\$Fon Disk\$Wa;	*Retry: ;	ed code s nitiates A Disk\$Hun H,Disk\$T B,Disk\$T Set\$Watc A,M A Disk\$Com Disk\$Hun A Disk\$Err Disk\$Wai	ets up c the disk g\$Flag imed\$Out imer hdcg g\$Flag or t\$Loop	;Re-ent isk con coperat ;Wait u ; oper ; if a ;On ent ; contro ; rout	ry point to retry after error troller command block ion ntil disk status block indicates ation has completed, then check ny errors occurred ry HL -> disk control byte ;Ensure hung flag clear ;Set up watchdog timer ;Time delay ;Get control byte ;Operation done ;Also check if timed out ;Will be set to 40H L arrives here from watchdog ine itself so this is effectively of the interrupt service routine

Figure 9-5. (Continued)

Disk\$Complete: ;Reset watchdog timer ;HL is irrelevant here 0037 010000 LXI в,о 003A CD3B03 CALL Set\$Watchdog ;Complete -- now check status Disk\$Status\$Block 003D 3A4300 I DA ;Check if any errors occurred 0040 FE80 CPT 80H JC Disk\$Error :Yes 0042 DA9F02 Disk#Error#Ignore: ;No 0045 AF 0046 320600 0049 C9 XRA ;Clear error flag Disk\$Error\$Flag STA RET ; ; Disk error message handling : Disk\$Error\$Messages: ;This table is scanned, comparing the ; disk error status with those in the table. Given a match, or even when the end of the table is reached, the address following the status value points to the correct advisory message text. Following this is the address of an ; error description message. 004A 40 DB 40H 004B B0019500 004F 41 nш Disk\$Advice1,Disk\$Msg\$40 DB 41H 0050 C9019A00 Disk\$Advice2, Disk\$Msg\$41 DW 0054 42 DB 42H 0055 E301A400 Disk\$Advice3,Disk\$Msg\$42 DW 0059 21 DB 21H 005A 0702B400 DW Disk\$Advice4,Disk\$Msg\$21 005E 22 005F 1B02B900 **NR** 22H nω Disk\$Advice5,Disk\$Msg\$22 0063 23 DB 23H 0064 1B02C000 Disk\$Advice5,Disk\$Msg\$23 DW 0068 24 0069 3D02D200 **DB** 24H DW Disk\$Advice6, Disk\$Msg\$24 006D 25 DB 25H 006E 3D02DE00 Disk\$Advice6,Disk\$Msg\$25 DW 0072 11 DB 11H 0073 5302F100 DW Disk\$Advice7,Disk\$Msg\$11 0077 12 DB 12H 0078 5302FF00 DW Disk\$Advice7,Disk\$Msg\$12 0070 13 DB 13H 0070 53020001 Disk\$Advice7, Disk\$Msg\$13 DW 0081 14 **DR** 14H Disk\$Advice7,Disk\$Msg\$14 0082 53021A01 ħω 0086 15 0087 53022901 DB 15H DW Disk\$Advice7, Disk\$Msg\$15 008B 16 DB 16H 008C 53023501 DW Disk\$Advice7,Disk\$Msg\$16 0090 00 DB ;<== Terminator 0091 53024501 Disk\$Advice7, Disk\$Msg\$Unknown ;Unmatched code DW 0005 = DEM\$Entry\$Size EQU 5 ;Entry size in error message table Message texts 3 DB 'Hung',0 ;Timeout message 0095 48756F6700Disk\$Msg\$40: 009A 4E6F742052Disk\$Msg\$41: 00A4 5772697465Disk\$Msg\$42: 'Not Ready',0 DB 'Write Protected',0 DB 00B4 4461746100Disk\$Msg\$21: 'Data',0 DB 00B9 466F726D61Disk\$Msg\$22: 'Format',O DB 00C0 4D69737369Disk\$Msg\$23: DB 'Missing Data Mark',0 00D2 4275732054Disk\$Msg\$24: DB 'Bus Timeout',0 'Controller Timeout',0 00DE 436F6E7472Disk\$Msg\$25: DB OOF1 4472697665Disk\$Msg\$11: DB 'Drive Address',0 'Head Address',0 OOFF 4865616420Disk\$Msg\$12: **DR** 'Track Address',0 010C 547261636BDisk\$Msg\$13: DB

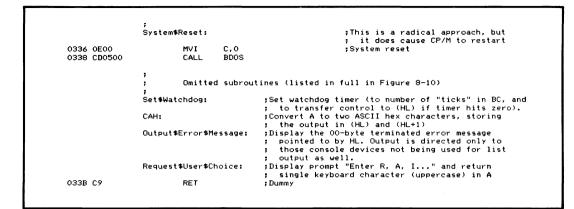
Figure 9-5. (Continued)

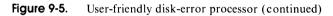
```
011A 536563746FDisk$Msg$14:
                                   DB
                                            'Sector Address',0
0129 4275732041Disk$Msg$15:
                                            'Bus Address',0
                                   DB
0135 496C6C6567Disk$Msg$16:
                                            'Illegal Command',0
                                   DB
                                            nв
0145 556E6B6E6FDisk$Msg$Unknown:
                                                     'Unknown', O
                 Disk$EM$1:
                                                     ;Main disk error message -- part 1
0140 070004
                                   DR
                                            BELL, CR, LF
0150 4469736B20
                                   DB
                                            (Disk 4.0)
                 .
                                                     ;Error text output next
                 .
Disk$EM$2:
                                                     ;Main disk error message -- part 2
0156 204572726F
                                            1 Error (
                                   ΠR
015E 0000
                 Disk$EM$Status:
                                   DB
                                            0,0
                                                     ;Status code in hex
0160 290D0A2020
                                            0 ;Disk drive code, A,B...
/, Head /
                                   DB
016E 00 1
016F 2C20486561
                 Disk$EM$Drive:
                                            o
                                   DR
                                   DB
0176 00
0177 2C20547261
                 Disk$EM$Head:
                                   DR
                                            0
                                                     ;Head number
                                            0,0 ;Track number
1, Sector 1
                                   DR
017F 0000
                Disk$EM$Track:
                                            0.0
                                   DB
0181 2020536563
                                   DR
                                                    Sector number
018A 0000 I
018C 2C204F7065
                Disk$EM$Sector:
                                            0,0
                                   DB
                                             , Operation -
                                   DB
019A 00
                                            ٥
                                   DR
                                                              :Terminator
019B 526561642EDisk$EM$Read:
                                   DB
                                            'Read.',0
'Write.',0
                                                              ;Operation names
01A1 5772697465Disk$EM$Write:
                                   DB
01A8 0D0A202020Disk$Advice0:
                                   np
                                            CR.LF.
                                                         1.0
                                            'Check disk loaded, Retry',0
01B0 436865636BDisk$Advice1:
                                   DB
01C9 506F737369Disk$Advice2:
                                   DB
                                            'Possible hardware problem',0
01E3 5772697465Disk$Advice3:
                                            'Write enable if correct disk, Retry',0
                                   DB
0207 5265747279Disk$Advice4:
021B 5265666F72Disk$Advice5:
                                            'Retry several times',0
'Reformat disk or use another disk',0
'Hardware error, Retry',0
                                   DB
                                   DB
023D 4861726477Disk$Advice6:
                                   DR
0253 4861726477Disk$Advice7:
                                   DB
                                            'Hardware or Software error, Retry'.0
0275 2C206F7220Disk$Advice9:
                                   DB
                                            ', or call for help if error persists',CR,LF
                 Disk$Action$Confirm:
029B 00
                                   DB
                                                    ;Set to character entered by user
                                            CR,LF,O
029C 0D0A00
                                   DR
                 ;
                          Disk error processor
                 1
                 .
                          This routine builds and outputs an error message.
                 2
                          The user is then given the opportunity to:
                 .
                 .
                                   R -- retry the operation that caused the error
                                   I -- ignore the error and attempt to continue
                 :
                                   A -- abort the program and return to CP/M
                 :
                 Disk$Error:
029F F5
                          PUSH
                                   PSW
                                                    ;Preserve error code from controller
s ;Convert code for message
02A0 215E01
                                   H,Disk$EM$Status
                          I X T
02A3 CD3B03
                          CALL
                                   CAH
                                                              ;Converts A to hex
0246 340700
                          LDA
                                   In$Buffer$Disk
                                                              :Convert disk id. for message
02A9 C641
                          ADI
                                    Δ '
                                                              ;Make into letter
02AB 326E01
                          STA
                                   Disk$EM$Drive
02AE 3A0200
02B1 C630
                          LDA
                                   Floppy$Head
                                                              ;Convert head number
                          ADI
                                   10
                                   Disk$EM$Head
02B3 327601
                          STA
02B6 3A0300
02B9 217F01
                         LDA
                                   Floppy$Track
                                                              ;Convert track number
                          LXI
                                   H, Disk$EM$Track
02BC CD3B03
                          CALL
                                   CAH
02BF 3A0400
                          LDA
                                   Floppy$Sector
                                                              :Convert sector number
02C2 218A01
                          LXI
                                   H, Disk$EM$Sector
02C5 CD3B03
                          CALL
                                   CAH
02C8 214D01
02CB CD3B03
                          IXT
                                   H,Disk$EM$1
                                                              ;Output first part of message
                          CALL
                                   Output$Error$Message
```

Figure 9-5. (Continued)

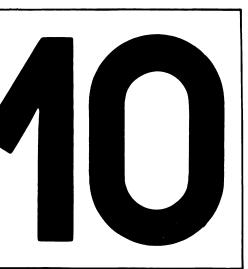
:Recover error status code 02CE F1 POP PSW 02CE 47 MOV B.A :For comparisons DEM\$Entry\$Size 02D0 214500 H,Disk\$Error\$Messages -IXT ;HL -> table -- one entry 02D3 110500 D.DEM\$Entry\$Size For loop below LXI Disk\$Error\$Next\$Code: D ;Move to next (or first) entry 02D6 19 DAD 02D7 7E MOV ;Get code number from table A.M 02D8 B7 ORA Check if end of table 02D9 CAE302 JZ Disk\$Error\$Matched ;Yes, pretend a match occurred 02DC B8 CMP ;Compare to actual code B 02DD CAE302 . 17 Disk\$Frror\$Matched ;Yes, exit from loop 02E0 C3D602 . IMP Disk\$Error\$Next\$Code ;Check next code Disk\$Error\$Matched: 02E3 23 ;HL -> advisory text address TNX н 02E4 5E MOV E,M 02E5 23 INX Ĥ 02E6 56 MOV D,M ;DE -> advisory test 02E7 D5 PUSH D ;Save for later 02E8 23 INX н ;HL -> message text address 02E9 5E MOV E,M ;Get address into DE 02EA 23 INX н 02EB 56 MOV D.M OPEC FR XCHG :HL -> text 02ED CD3B03 CALL Output\$Error\$Message Display explanatory text H, Disk\$EM\$2 ;Display second part of message 02F0 215601 LXI 02F3 CD3B03 Output\$Error\$Message CALL 02F6 219B01 LXI H,Disk\$EM\$Read ;Choose operation text (assume a read) 02F9 3A0100 LDA Floppy\$Command ;Get controller command 02FC FE01 CPI, Floppy\$Read\$Code 02FE CA0403 JZ Disk\$Error\$Read • Ve e :No. change address in HL 0301 21A101 LXI H,Disk\$EM\$Write Disk\$Frror\$Read: 0304 CD3B03 CALL Output\$Error\$Message ;Display operation type 0307 21A801 LXI H,Disk\$Advice0 ;Display leading blanks 030A CD3B03 CALL Output\$Error\$Message 030D E1 POP ;Recover advisory text pointer 030F CD3B03 Output\$Error\$Message CALL 0311 217502 IXI H.Disk\$Advice9 Display trailing component 0314 CD3B03 Output\$Error\$Message CALL . Disk\$Error\$Request\$Action: ;Ask the user what to do next Request\$User\$Choice ۰. 0317 CD3B03 CALL ; Display prompt and get single ; character response (folded to uppercase) CPI 031A FE52 **1**R1 ;Retry 031C CA2C03 031F FE41 Disk\$Error\$Retry .17 CPI : Abort? ´Α 0321 CA3603 JZ System\$Reset 0324 FE49 CPI 11 : Ignore? 0326 CA4500 JZ Disk\$Error\$Ignore 0329 C31703 JMP Disk\$Error\$Request\$Action Disk\$Error\$Retry: ;The decision on where to return to depends on whether the operation . failed on a deblocked or nondeblocked drive 032C 3A0500 032F B7 I DA Deblocking\$Required ORA 0330 C21500 Deblock\$Retry . IN7 0333 C30800 JMF No\$Deblock\$Retry

Figure 9-5. (Continued)





Basic Debugging Techniques Debug Subroutines Software Tools for Debugging Bringing Up CP/M for the First Time Debugging the CP/M Bootstrap Loader Debugging the BIOS Live Testing a New BIOS



Debugging A New CP/M System

This chapter deals with some of the problems you will face bringing up CP/M on a computer system for the first time or enhancing it once it is up and running on your system.

In the first case, when CP/M does not yet run on your computer, you may be writing the complete BIOS yourself, although you can model what you do on the example BIOS provided on the CP/M release diskette and the example code from Chapter 6.

In the second case, you can extend the existing BIOS by adding code — from the examples in Chapters 8 and 9, code from computer magazines, or code you create yourself. To do this, you will need access to the BIOS source code — a problem if the manufacturer of your computer does not make it available. In general, however, the BIOS source code is included with the system or can be obtained at nominal or no cost. If you cannot obtain the source code, you can, of course, take the bull by the horns and reimplement CP/M on your system. This may require many hours of disassembling the current BIOS machine code to find out how to access all the various ports and how to control the devices to which they are connected.

Although the BIOS is the major component of a new CP/M implementation, remember that it is only the beginning—you can spend the same amount of time and effort getting the bootstrap loader and all the utilities to function.

Basic Debugging Techniques

Before getting involved in the details of how to debug a CP/M implementation, it is worth considering the nature of the task. Some quotations that are appropriate here:

"Program testing can be used to show the presence of bugs, but never to show their absence." — Dijkstra

"We call them bugs because to call them mistakes would be psychologically unacceptable." — Hopkins

"Constants aren't, variables won't."

-Osborne

Debugging is the name we give to the process of executing programs and ascertaining whether the programs are running correctly. "Correctly" means in accordance with the mental model we have built of how the program should behave, subject to the constraints imposed by the physical hardware. Therein lies the first of the problems; you and the hardware are the arbiters of correct performance. The hardware is usually unforgiving; if there is a flaw in the way you program it, it will either be dramatically "uncooperative" or not work at all. As for how you perceive the system, several fairly simple tests, along with attempts to use the system for useful work for a few days, will shake the system down fairly well. The most difficult problems will be with intermittent failures or logical contradictions.

Computers are deterministic. That is, if you start from a known state and perform a known series of operations, the computer will always yield the same results. To achieve a known state is not so difficult—resetting the system and clearing memory will do it. Performing a known series of operations just means running the program again, although if you are using interrupts, you cannot truthfully say that exactly the same operations are being performed, because the interrupts will not happen at *exactly* the same time as before.

The "Orville Wright" Approach

Your role in debugging a new CP/M system is comparable to the popular, though untrue, idea of the way the Wright brothers developed flying machines:

build a machine, take it to the top of a hill, throw it off, and, when it crashes, examine the debris to discover what went wrong.

Each time you do an assembly and test, you are building the aircraft and lobbing it off the edge of a cliff. Each time it crashes, you examine the wreckage and try to determine the possible cause.

This is a highly inferential process. With the wreckage as a starting point, you use inference and intuition to extrapolate the real problem and the correction for it.

Built-In Debug Code

The single most important concept that you will need in testing CP/M systems is the same as that used in the modern day "black box" flight recorder. This device is essentially a multi-channel tape recorder that records all of the relevant conditions of the aircraft, its height, altitude, throttle settings, flap settings, and even the voice communications among crew members. If the airplane crashes, investigators can replay the information and understand what happened during the flight.

Applying this concept to debugging CP/M means that you must build into your code some method for recording what it is doing, so that if the system crashes, you can see what it was doing. Make the code tell you what went wrong.

The debug code should be designed at the same time as the rest of the program. Plan the debugging code while the design is still on the drawing board. The source code for debugging should be a permanent part of the BIOS. Use conditional assembly to "IF" out most of the debug code from the final version, or make the code sensitive to a flag in the configuration block so that you can re-enable the debug code at a moment's notice if the system begins to behave strangely.

The more meaningful the debug output data, the less you will have to guess at what is wrong, and therefore the less painful and time-consuming the debugging process will be. Make the output intelligible to others who may use it or yourself several months hence. Data that tells you what is happening is more useful than internal hexadecimal values, particularly if someone else must interpret it or relay it to you over the telephone.

Debug Subroutines

Many programmers do their debugging on a casual "catch as catch can" basis because they are overwhelmed by the task of building the necessary tools. Others are too eager to start on a new program to take a few extra hours or days to build debug subroutines.

To help solve this problem, the following section provides some ready-made debugging tools that can be used "as is." Each of these routines has been thor-

322 The CP/M Programmer's Handbook

oughly debugged (there's nothing worse than debug code with bugs in it!) and has been used in actual program testing.

Overall Design Philosophy

Some common methods run through the examples that follow. These include displaying meaningful "captions" (including the specific address that called the debug routine), grouping all debugging code together, preserving the contents of all registers, and setting up the stack area in a standard way.

Debug Code Captions When the contents of registers or memory are output as part of a debugging process, a caption of explanatory text describing the values should be displayed. For example, rather than displaying the contents of the A register like this,

A = 1F

you can use a meaningful caption such as:

Transaction Code A = 1F.

.

When you write additional debugging code, especially if you need to add it to an existing routine, it is cumbersome to have to write the call to the debug routine and then search through the source code to find a convenient place to put an ASCII caption string. A caption string several pages removed from the point where it is referenced makes for problems when you want to relate the debug output on the screen or listing to the source code itself. Therefore, all of the routines that follow allow you to declare the caption strings "in-line" like this:

```
IF DEBUG
CALL Debug$Routine
DB 'Caption string here',CR,LF,O
ENDIF
```

MVI

All of the following routines that output a caption recognize one specific 8-bit value in the caption string. If they encounter a value of 0ADH (mnemonic for ADdress), they will output the address of the byte following the call to the debug routine. For example,

;Next instruction

0210 CALL Debug\$Routine 0213 DB 0ADH, Caption string1,0

will cause the routine to display the following:

0213 Caption string

This identifies the point in your program from which the debug routine was called, and thus avoids any possible ambiguity between different calls to the same debug routine with similar captions.

Grouping Debug Code Grouping all the debug code together lends itself to using conditional assembly with IF/ENDIF statements.

Setting Up the Stack Area All of the following routines preserve the CPU registers so that there are no side effects from using them. All of them assume that they can use the stack pointer and that there is sufficient room in the stack area. Hence you will need to declare adequate stack space for your main code and for the debug routines. Fill the stack area with a known pattern like this:

Then, during debugging, you can examine the stack area and determine how much of it is unused. For example, if you looked at the stack area you might see something like this:

Stack area overflow can give arcane bugs; the program seems to leap off into space in a nondeterministic way. By setting up the stack area in this way, you can recognize an overflow condition easily.

Debug Initialization Before you can execute any of the debug subroutines in this chapter, you must make a call to the initialization subroutine, DB\$Init. The DB\$Init routine sets up some of the internal variables needed by the debug package. You may need to add some of your own initialization code here.

Console Output

Normally, you can use the CONOUT functions either via the BDOS (Function 2), or via the BIOS by calling the jump vector directly. You cannot do this when you need to debug console routines themselves, nor when you need to debug interrupt service routines. In the latter case, if an interrupt pulled control out of the CONOUT routine in the BIOS, you would get unwanted re-entrancy if the debug code again entered the CONOUT driver to display a caption. Therefore, the debug routines have been written to call their own local CONOUT routine, which is called DB\$CONOUT. DB\$CONOUT can be changed to call the BDOS, the BIOS, or a "private" polled output routine.

A counterpart DB\$CONIN routine for console input is provided for essentially the same reasons.

-

Controlling Debug Output

All output of debug routines in this chapter is controlled by a single master flag, DB\$Flag. If this flag is nonzero, debug output will occur; if zero, all output is suppressed.

This flag can be set and cleared from any part of the program you are testing. It is especially useful when you need to debug a subroutine that is called many times from many different places. You can write additional code to enable debug output when certain conditions prevail; for example, when a particular track or sector is about to be written or when a character input buffer is almost full.

Two subroutines, DB\$On and DB\$Off, are shown that access the debug control flag. These, as their names suggest, turn debug output on and off.

Turning the debug output on and off from within the program can create a confusing display of debug output, lacking any apparent continuity. DB\$Off gives you the option of outputting a character string indicating that debug output has been turned off.

Pass Counters

Another method of controlling debug output is to use a *pass counter*, enabling debug output only after control has passed through a particular point in the code a specific number of times.

Two subroutines are provided for this purpose. DB\$Set\$Pass sets the pass counter to a specific value. DB\$Pass decrements this pass count each time control is transferred to it. When the pass count hits zero, the debug control flag DB\$Flag is nonzero and debug output begins.

Using pass counter techniques can save you time and effort in tracking down a problem that occurs only after the code has been running for several minutes.

Displaying Contents of Registers and Memory

Figure 10-2 shows a series of display subroutines, the primary one of which is DB\$Display. It takes several parameters, depending on the information you want displayed. The generic call to DB\$Display is as follows:

CALL	DB\$Display
DB	Code <- Indicates the data to be
	displayed
{DW	Optional additional parameters}
DB	'Caption string',0

The codes that can be used in this call are shown in Table 10-1.

The only function that uses additional parameters is DB\$Memory. This displays bytes from memory in hexadecimal and ASCII, using the start and finish addresses following the call. Here is an example:

CALL	DB\$Display
DB	DB\$Memory
DW	Start\$Address,End\$Address
DB	'Caption string',0

Table 10-1.Codes for DB\$Display

~

Code	Value displayed
	8-bit registers
DB\$F	Condition Flags
DB\$A	Register A
DB\$B	Register B
DB\$C	Register C
DB\$D	Register D
DB\$E	Register E
DB\$H	Register H
DB\$L	Register L
	Memory
DB\$Memory	Bytes starting and ending at the addresses specified by the two word values following the code value.
	16-bit registers
DB\$BC	Register pair BC
DB\$DE	Register pair DE
DB\$HL	Register pair HL
DB\$SP	Stack Pointer
	Byte values
DB\$B\$BC	Byte addressed by BC
DB\$B\$DE	Byte addressed by DE
DB\$B\$HL	Byte addressed by HL
	Word values
DB\$W\$BC	Word addressed by BC
DB\$W\$DE	Word addressed by DE
DB\$W\$HL	Word addressed by HL

Debugging Program Logic

In addition to displaying the contents of registers and memory, you need to display the program's execution path, not in terms of addresses, but in terms of the *problem*. You can do this by displaying debug messages that indicate what decisions have been made by the program as it executes. For example, if your BIOS checks a particular value to see whether the system should read or write on a particular device, the debug routine should display a message like this:

Entering Disk Read Routine

This is more meaningful than just displaying the function code for the drivers — although you may want to display this as well, in case it has been set to some strange value.

Two subroutines are provided to display debug messages. They are DB\$MSG and DB\$MSGI. Both of these display text strings are terminated with a byte of 00H. You can see the difference between the two subroutines if you examine the way they are called.

DB\$MSG is called like this:

```
LXI H,Message$Text ;HL -> text string
CALL DB$MSG
```

DB\$MSGI is called like this:

CALL DB\$MSG DB ODH,OAH,'Message Text',O ;In-line

DB\$MSGI is more convenient to use. If you decide that you need to add a message, you can declare the message immediately following the call. This also helps when you look at the listing, since you can see the complete text at a glance.

Use DB\$MSG when the text of the message needs to be selected from a table. Get the address of the text into HL and then call DB\$MSG to display it.

Creating Your Own Debug Displays

If you need to build your own special debug display routines, you may find it helpful to incorporate some of the small subroutines in the debug package. The following are the subroutines you may want to use:

DB\$CONOUT

Displays the character in the C register.

DB\$CONIN

Returns the next keyboard character in A.

DB\$CONINU

Returns the next keyboard character in A, converting lowercase letters to uppercase.

DB\$DHLH

Displays contents of HL in hexadecimal.

DB\$DAH

Displays contents of A in hexadecimal.

DB\$CAH

Converts contents of A to hexadecimal and stores in memory pointed at by HL.

DB\$Nibble\$To\$Hex

Converts the least significant four bits of A into an ASCII hexadecimal character in A.

DB\$CRLF

Displays a CARRIAGE RETURN/LINE FEED.

DB\$Colon

Displays the string ": ".

DB\$Blank

Displays a single space character.

DB\$Flag\$Save\$On

Saves the current state of the debug output control flag and then sets the flag "on" to enable debug output.

DB\$Flag\$Restore

Restores the debug output control flag to the state it was in when the DB\$Flag\$Save\$On routine was last called.

DB\$GHV

Gets a hexadecimal value from the keyboard, displaying a prompt message first. From one to four characters can be specified as the maximum number of characters to be input.

DB\$A\$To\$Upper

If the A register contains a lowercase letter, this converts it to an uppercase letter.

Debugging I/O Drivers

Debugging low-level device drivers creates special problems. The major one is that you do not normally want to read and write via actual hardware ports while you are debugging the code —either because doing so would cause strange things to happen to the hardware during the debugging, or because you are developing and debugging the drivers on a system different from the target hardware on which the drivers are to execute.

Before considering the solution, remember that the input and output instructions (IN and OUT) are each two bytes long. The first byte is the operation code (0DBH for input, 0D3H for output), and the second byte is the port number to "input from" or "output to."

Debug subroutines are provided here to intercept all IN and OUT instructions, displaying the port number and either accepting a hexadecimal value from the console and putting it into the A register (in the case of IN), or displaying the contents of the A register (for the OUT instruction).

IN and OUT instructions can be "trapped" by changing the operation code to one of two RST (restart) instructions. An RST is effectively a single-byte CALL instruction, calling down to a predetermined address in low memory. The debug routines arrange for JMP instructions in low memory to receive control when the correct RST is executed. The code that receives control can pick up the port number, display it, and then accept a hex value for the A register (for IN) or display the current contents of the A register (for OUT). The example subroutines shown later in this chapter use RST 4 in place of IN instructions, RST 5 for OUT.

Wherever you plan to use IN, use the following code:

IF	Debug 🕔
RST	4
ENDIF	
IF	NOT Debug
DB	IN
ENDIF	
DB	Port\$Number

Note that you can use the IN operation code as the operand of a DB statement. The assembler substitutes the correct operation code.

Use the following code wherever you need to use an OUT instruction:

IF	Debug
RST	5.
ENDIF	
IF	NOT Debug
DB	OUT
ENDIF	
DB	Port\$Number

When the RST 4 (IN) instruction is executed, the debug subroutine displays

1AB3 : Input from Port 01 : _

The "1AB3" is the address in memory of the byte containing the port number. It serves to pinpoint the IN instruction in memory. You can then enter one or two hexadecimal digits. These will be converted and put into the A register before control returns to the main program at the instruction following the byte containing the port number.

When the RST 5 (OUT) instruction is encountered, the debug subroutine displays

1AB5 : Output to Port 01 : FF

This identifies where the OUT instruction would normally be as well as the port number and the contents of the A register when the RST 5 (OUT) is executed.

Debugging Interrupt Service Routines

You can use a technique similar to that of the RST instruction just described to "fake" an interrupt. You preset the low-memory address for the RST instruction you have chosen for the jump into the interrupt service routine under test.

When the RST instruction is executed, control will be transferred into the interrupt service routine just as though an interrupt had occurred. You will need to intercept any IN or OUT instructions as described above—otherwise the code probably will go into an endless loop.

Before executing the RST instruction to fake the interrupt, load all the registers with known values. For example:

MVI	A,OAAH		
LXI	B,OBBCCH		
LXI	D,ODDEEH		
LXI	H,01122H		
RST	6	;Fake	interrupt
NOP			

When control returns from the service routine, you can check to see that it restored all of the registers to their correct values. An interrupt service routine that does not restore all the registers can produce bugs that are very hard to find.

Check, too, that the stack pointer register has been restored and that the service routine did not require too many bytes on the stack.

You also can use the CALL instruction to transfer control to the interrupt service routine in order to fake an interrupt. RST and CALL achieve the same effect, but RST is closer to what happens when a real interrupt occurs. As it is a single-byte instruction, it also is easier to patch in.

Subroutine Listings

Figure 10-1 is a functional index to the source code listing for the debug subroutines shown in Figure 10-2. The listing's commentary defines precisely how each debug subroutine is called.

Figure 10-3 shows the output from the debug testbed.

Software Tools for Debugging

In addition to building in debugging subroutines, you will need one of the following proprietary debug programs:

DDT (Dynamic Debugging Tool)

This program, included with the standard CP/M release, allows you to load programs, set and display memory and registers, trace through your program instruction by instruction, or execute it at full speed, but stopping

Start Line	Functional Component or Routines
00001	Debug subroutine's Testbed
00100	Test register display
00200	Test memory dump display
00300	Test register pair display
00400	Test byte indirect display
00500	Test DB\$On/Off
00600	Test DB\$Set\$Pass and DB\$Pass
00700	Test debug input/output
00800	Debug subroutines themselves
01100	DB\$Init - initialization
01200	DB\$CONINU - get uppercase keyboard character
01300	DB\$CONIN - get keyboard character
01400	DB\$CONOUT - display character in C
01500	DB\$On - enable debug output
01600	DB\$Off - disable debug output
01700	DB\$Set\$Pass - set pass counter
01800	DB\$Pass - execute pass point
01900	DB\$Display - main debug display routine
02200	Main display processing subroutines
02500	DB\$Display\$CALLA - display CALL's address
02600	DB\$DHLH - display HL in hexadecimal
02700	DB\$DAH - display A in hexadecimal
02800	DB\$CAH - convert A to hexadecimal in memory
02900	DB\$Nibble\$To\$Hex - convert LS 4 bits of A to hex.
02930	DB\$CRLF - display Carriage Return, Line Feed
02938	DB\$Colon - display " : "
02946	DB\$Blank - display ""
03100	DB\$MSGI - display in-line message
03147	DB\$MSG - display message addressed by HL
03300	DB\$Input - debug INput routine
03500	DB\$Output - debug OUTput routine
03700	DB\$Flag\$Save\$On - save debug flag and enable
03800	DB\$Flag\$Restore - restore debug control flag
03900	DB\$GHV - get hexadecimal value from keyboard
04100	DB\$A\$To\$Upper - convert A to upper case

Figure 10-1.	Functional	index	for	Figure	10-2
--------------	------------	-------	-----	--------	------

at certain addresses (called breakpoints). It also has a built-in miniassembler and disassembler so you do not have to hand assemble any temporary code "patches" you add.

SID (Symbolic Interactive Debug)

Similar to DDT in many ways, SID has enhancements that are helpful if you use Digital Research's MAC (Macro Assembler) or RMAC (Relocating Macro Assembler). Both of these assemblers can be told to output a file

00001								
00002								
00003	;							
00004	;	Dehug	Subroutines					
00005	,	Denda						
00006	;<		NOTE:					
00007	;		The line numbers at the extreme left are included purely					
	;	to ret	to reference the code from the text.					
00009	7	Inere	There are deliberately induced discontinuities					
00010	;	in the	in the numbers in order to allow space for expansion.					
00011	;	_						
00012	7			these routines thoroughly,				
00013	;			any changes, the testbed				
00014	;			e itself has been left in				
00015	;	in thi	is figure.					
00016	;							
00017	;	Debug	testbed					
00018	;							
00019	0100	ORG	100H					
00020	STAR	Τ:						
00021	0100 316B03	LXI	SP, Test\$Stack	;Set up local stack				
00022	0103 CDEA04	CALL	DB\$Init	;Initialize the debug package				
00023	0106 CD1505	CALL	DB\$On	;Enable debug output				
00023	1100 001000	ONCE		;Simple test of A register display				
00024	0109 3EAA	MVI	A, OAAH	;Simple test of A register display ;Preset a value in the A register				
00026	010B 01CCBB	LXI	B, OBBCCH	Prefill all other registers, partly				
00027	010E 11EEDD	LXI	D, ODDEEH	; to check the debug display, but				
00028	0111 2111FF	LXI	H,OFF11H	; also to check register save/restore				
00100	; #							
00101	;	Test r	egister display					
00102	;							
00103	0114 B7	ORA	A	;Set M-flag, clear Z-flag, set E-flag				
00104	0115 37	STC		;Set carry				
00105	0116 CD5205	CALL	DB\$Display	;Call the debug routine				
00106	0119 00	DB	DB\$F					
00107	011A 466C616773	DB	'Flags',0					
00108	;							
00109	0120 CD5205	CALL	DB\$Display	;Call the debug routine				
00110	0123 02	DB	DB\$A	,				
00111	0124 4120526567	DB	'A Register',0					
00112	· · · · · · · · · · · · · · ·	00	H Register ;0					
00113	012F CD5205	CALL	DB\$Bisplay	•Call the debug routine				
00113	0132 04	DB	DB#DISPIAY DB\$B	;Call the debug routine				
			2242					
00115	0133 4220526567	DB	'B Register',O					
00116	;							
00117	013E CD5205	CALL	DB\$Display	;Call the debug routine				
00118	0141 06	DB	DB\$C					
00119	0142 4320526567	DB	<pre>^C Register',0</pre>					
00120	;							
00121	014D CD5205	CALL	DB\$Display	;Call the debug routine				
00122	0150 08	DB	DB\$D					
00123	0151 4420526567	DB	'D Register',O					
00124	;							
00125	015C CD5205	CALL	DB\$Display	;Call the debug routine				
00126	015F 0A	DB	DB\$E	-				
00127	0160 4520526567	DB	'E Register',0					
00128	•••••	22	E Register ,o					
00129	016B CD5205	CALL	DB\$Display	;Call the debug routine				
00130	0165 00	DB	DB\$H	ycall the debug routine				
00131	016E OC 016F 4820526567	DB	'H Register',0					
00132		00	in negister ,0					
	0174 CDE20E	C 41 1						
00133	017A CD5205 017D OE	CALL	DB\$Display	;Call the debug routine				
00134		DB	DB\$L					
00135	017E 4C20526567	DB	'L Register',O					
00200	;#							
00201	;	Test №	lemory Dump Display					
00202	;							
00203	0189 CD5205	CALL	DB\$Display					
00204	018C 18	DB	DB\$M	;Dump memory				
00205	018D 08012801	DW	108H, 128H	;Check start/end at nonmultiples				
00206	0191 4D656D6F72	DB	'Memory Dump #1',0	; of 10H				
00207	;							
00208	01A0 CD5205	CALL	DB\$Display					
00209	01A3 18	DB	DB\$M	;Dump memory				
00210	01A4 00011F01	DW	100H, 11FH	;Check start and end on displayed				
00211	01A8 4D656D6F72	DB	'Memory Dump #2',0	; line boundaries				
00212	01H0 4000000172	~~		, Lane overleer as a				
	,							

Figure 10-2. Debug subroutines

00213	01B7 CD5205	CALL	DB\$Display	
00213	01BA 18	DB	DB\$M	;Dump memory
00215	01BB 01010001	DW	101H. 100H	Check error handling where
00215	01BF 4D656D6F72	DB	'Memory Dump #3',0	; start > end address
		UB	Hemory Dump #3,0	; start / enu auuress
00217	, ,	CALL	DB\$Display	
00218	01CE CD5205			- Dume memory
00219	01D1 18	DB	DB\$M	; Dump memory
00220	01D2 00010001	DW	100H, 100H	;Check end-case of single byte
00221	01D6 4D656D6F72	DB	'Memory Dump #4',0	; output
00300	- ;#			
00301	;	Test r	egister pair display	
00302	, , , , , , , , , , , , , , , , , , , ,			
00303	01E5 CD5205	CALL	DB\$Display	;Call the debug routine
00304	01E8 10	DB	DB\$BC	
00305	01E9 4243205265	DB	'BC Register',O	
00306	;			
00307	01F5 CD5205	CALL	DB\$Display	;Call the debug routine
00308	01F8 12	DB	DB\$DE	
00309	01F9 4445205265	DB	'DE Register',O	
00310	;			
00311	0205 CD5205	CALL	DB\$Display	;Call the debug routine
00312	0208 14	DB	DB\$HL.	
00313	0209 484C205265	DB	'HL Register',O	
00314	;			
00315	0215 CD5205	CALL	DB\$Display	;Call the debug routine
00316	0218 16	DB	DB\$SP	
00317	0219 5350205265	DB	'SP Register',0	
00318	;			
00319	0225 013203	LXI	B,Byte\$BC	;Set up registers for byte tests
00320	0228 113303	LXI	D,Byte\$DE	
00321	022B 213403	LXI	H,Byte\$HL	
00400	;#			
00401	,	Test b	yte indirect display	
00402	í			
00403	022E CD5205	CALL	DB\$Display	;Call the debug routine
00404	0231 14-	DB	DB\$B\$BC	· · · · · · · · · · · · · · · · · · ·
00405	0232 4279746520	DB	'Byte at (BC)',0	
00406	1	20	D , te a t (DC) , 0	
00407	023F CD5205	CALL	DB\$ Display	;Call the debug routine
00408	0242 10	DB	DB\$B\$DE	,call the debug foutine
00409	0243 4279746520	DB	'Byte at (DE)',0	
00407	0243 42/7/48320	55	byte at (DE) ,0	
00410	7			
00411	0250 CD5205	CALL	DB\$Display	;Call the debug routine
00412	0253 1E	DB	DB\$B\$HL	
00413	0254 4279746520	DB	'Byte at (HL)',O	
00414	;			
00415	0261 013503	LXI	B,Word\$BC	;Set up the registers for word tests
00416	0264 113703	LXI	D,Word\$DE	
00417	0267 213903	LXI	H,Word\$HL	
00418				
00419	026A CD5205	CALL	DB\$Display	;Call the debug routine
00420	026D 20	DB	DB\$W\$BC	-
00421	026E 576F726420	DB	'Word at (BC)',0	
00422	;			
00423	027B CD5205	CALL	DB\$Display	;Call the debug routine
00424	027E 22	DB	DB\$W\$DE	_
00425	027F 576F726420	DB	'Word at (DE)',0	
00426	;			
00427	028C CD5205	CALL	DB\$Display	;Call the debug routine
00428	028F 24	DB	DB\$W\$HL	
00429	0290 576F726420	DB	'Word at (HL)',0	
00500	;#			
00501	; "	Test I)B\$On/Off	
00502	;			
00503	029D CD1D05	CALL	DB\$Off	;Disable debug output
00504	02A0 CDD607	CALL	DB\$MSGI	;Display in-line message
00505	02A3 0D0A546869	DB		e should NOT appear',0
00506	02H0 000H040007	55	And A Court A LITE MERRIA	e sueere net affect 14
00507	02C4 CD1505	CALL	DB\$On	
00508	02C7 CDD607	CALL	DB\$MSGI	
00509	02CA 0D0A446562	DB		t has been re-enabled.′,0
00600	02CH 0D0H448382 ;#	20	control beday output	· ···· ·······························
00600	; *	Toct .	ass count logic	
00602	; ;		ass count rogic	
00002	*			

00603		CD1D05		CALL	DB\$Off		;Disable debug output
00604		CD2405		CALL	DB\$Set\$	Pass	;Set pass count
00605	02F4	1E00		DW	30		
00606			;				
00607	02F6	3E22		MVI	A,34		;Set loop counter greater than pass
00608							; counter
00609			Test\$Pa	ss\$Loop:			
00610	02F8	CD3505		CALL	DB\$Pass		;Decrement pass count
00611		CDD607		CALL	DB\$MSG I		;Display in-line message
00612		0D0A54686	7	DB			nould display 5 times',0
00613	0324			DCR	A	,	
00614		C2F802		JNZ	Test\$Pa	ss\$Loop	
00700			;#				
00701			;	Test de	bug inpu	t/output	
00702			;				
00703	0328	CD1D05	,	CALL	DB\$Off		;Check that debug IN/OUT
00704				0.122			; must still occur when debug
00705							; output is disabled.
00706	032B	F7		RST	4		; Debug input
00707	0320			DB	11H		;Port number
00708	0320	EE .		RST	5		;Debug output (value return from input)
00709	032D 032E	22		DB	22H		;Debug output (Value return from input) ;Port number
00710				22			TOT CHUNCT
00711	032F	C30000		JMP	0		;Warm boot at end of testbed
00712	UCE		;	0.11	÷		ywarm boot at end of testbed
00713			;				
00714			;		alues for	r byte and word o	ticolave
00715	0332	BC	; Byte\$BC:		DB	r byte and word t OBCH	11341413
00716	0332		Byte\$DE:		DB	ODEH	
00717	0334		Byte\$HL:		DB	OF1H	
00718	0334			•	56	VI 10	
00719	0225	OCOB	; Word\$BC:		DW	OBOCH	
00720		OEOD	Word\$DE:		DW	OBOEH	
00721		010F	Word\$HL:		DW	0F01H	
00722	0557	0100	woru write	•	DW	OFUTH	
00723	0228	99999999999			DW	00000 00000 0000	
00723		99999999999			DW		9H, 9999H, 9999H, 9999H, 9999H, 9999H
00725					DW		9H, 9999H, 9999H, 9999H, 9999H, 9999H
00725	0328	99999999999		1	DW	9999H, 9999H, 9995	эн, 9999н, 9 999н, 9999н, 9999н, 9999н
			Test\$Sta	ACK:			
00727			;				
00728			;				
00729			;				
00730	0400			ORG	400H		;To avoid unnecessary listings
00731							; when only the testbed changes
00732			;				
00800			;#				
00801			;				
00802			;	Debug s	ubroutine	25	
00803			;				
00804			;				
00805			;			Display codes	
00806			;				own the table of addresses
00807			;	for var:	ious subr	routines to be us	sed.
00808			;				
00809	0000		DB\$F	EQU	00	;Flags	
00810	0002		DB\$A	EQU	02	;A register	
00811	0004		DB\$B	EQU	04	; B	
00812	0006		DB\$C	EQU	06	;C	
00813	0008		DB\$D	EQU	08	; D	
00814	000A	=	DB\$E	EQU	10	;E	
00815	000C	=	DB\$H	EQU		; H	
00816	000E		DB\$L	EQU		; L	
00817	0010		DB\$BC	EQU	16	; BC	
00818	0012		DB\$DE	EQU	18	; DE	
00819	0014		DB\$HL	EQU	20	;HL	
00820	0016		DB\$SP	EQU	22	Stack pointer	
00821	0018	=	DB\$M	EQU	24	; Memory	
00822	001A		DB\$B\$BC	EQU	26	;(BC)	
00823	001C	=	DB\$B\$DE		28	;(DE)	
00824	001E		DB\$B\$HL		30	;(HL)	
00825	0020	=	DB\$W\$BC	EQU	32	;(BC+1),(BC)	
00826	0022		DB\$W\$DE	EQU	34	;(DE+1),(DE)	
00827	0024		DB\$W\$HL		36	;(HL+1),(HL)	
			;			-	
1 00828							
00828			•				
00829			;	Fauster			
00829 00830	0020		1	Equates	FOL	20H •Address	for RST 4 - IN instruction
00829	0020		;	Equates	EQU	20H ;Address	; for RST 4 - IN instruction

Figure 10-2. (Continued)

00832	0028	-	RST5	EQU	28H	Address for RST 5 - OUT instruction
00833			1			
00834	0001			EQU	1	;BDOS CONIN function code
00835	0002		B\$CONOUT	EQU	2	BDOS CONOUT function code
00836	000A		B\$READCONS	EQU	10	;BDOS read console function code
00837	0005	-	BDOS	EQU	5	;BDOS entry point
00838			<u>;</u>		-	
00839	0000			EQU	0	
00840	FFFF '	=		EQU	NOT Fals	e .
00841			;			
00842						;Equates to specify how DB\$CONOUT
00843						; and DB\$CONIN should perform
00844	0000	_		EQU	Enler	; their input/output
00845	0000			EQU	False False	;) :) Only one must be true
00846	FFFF			EQU	False True	;) Only one must be true ;)
00847	CEFE		1 DR#RDO2#10	EQU .	11.05	i /
			,			;Equates for polled I/O
00849	0001	-	DB\$Status\$Port	EQU	01H	;Equates for polled 1/0 ;Console status port
00850	0002			EQU	02H	Console data port
00852	0002	-		EGO	ven	, console data port
00852	0002	-	; DB\$Input\$Ready	EQU	0000\$001	OB ;Incoming data ready
00854	0002		DB\$Output\$Ready		0000\$000	
00855	0001		:	200		ab yneddy fol odipat
00856			,			;Data for BIOS I/O
00857	0400	сз	BIOS\$CONIN:	DB	JMP	;The initialization routine sets these
00858	0401			DW	0	; two JMP addresses into the BIOS
00859	0403		BIOS\$CONOUT:	DB	JMP	,
00860	0404			DW 1	0	
00861	/		,			
00862				ug varia	ables and	constants
00863			;			
00864	0406	00	DB\$Flag:	DB	0	;Main debug control flag
00865			-			; When this flag is nonzero, all debug
00866						; output will be made. When zero, all
00867						; debug output will be suppressed.
00868						; It is altered either directly by the user
00869						; or using the routines DB\$On, DB\$Off and
00870						; DB\$Pass.
00871			;			
00872	0407	0000	DB\$Pass\$Count:	DW	0	;Pass counter
00873						; When this is nonzero, calls to DB\$Pass
00874						; decrement it by one. When it reaches
00875						; zero, the debug control flag, DB\$Flag,
00876						; is set nonzero, thereby enabling
00877						; debug output.
00878			1			- O
00879			DB\$Save\$HL:		~	;Save area for HL
00880	0409		DB\$Save\$L:	DB	0	
00881	040A	00	DB\$Save\$H:	DB	0	
00882				-	•	· Course for stack saister
00883	040B		DB\$Save\$SP:	DW	0	;Save area for stack pointer
00884	040D 040F	0000	DB\$Save\$RA:	DW	0 пш	;Save area for return address 0 ;Starts out the same as DB\$Save\$RA
00885	0401	0000	DB\$Call\$Address:		DW	
00886						
00888			BD#C+ sy+#Addu			; output ahead of the caption ;Start address for memory display
00889	0411	0000	DB\$Start\$Address	DW	0	, orail address for memory display
00890	0411	0000	DP#End#Address.	L'W	0	;End address for memory display
00891	0413	0000	DB\$End\$Address:	DW	0	FCUD ADDLESS TOL WEMOLA DISPIN
	0413	0000	DR&Dicolay#Codes		5	;Display code requested
00893	0415	00	DB\$Display\$Code:	DB	0	Jurspra, come lequesten
00895	0410	~~	;		~	
00896			,			
00897			,			;Stack area
00898	0416	9999999999	,	DW	9999H.9	, Stack al ea
00899		99999999999		DW		ууун, уууун, уууун, уууун, уууун, уууун, уууун
00900		99999999999		DW		ууун, уууун, уууун, уууун, уууун, уууун, уууун
00901	0446		DB\$Save\$E:	DB	0	;E register
00902	0447		DB\$Save\$D:	DB	ŏ	;D register
00903	0448		DB\$Save\$C:	DB	õ	;C register
00904	0449		DB\$Save\$B:	DB	õ	;B register
00905	044A		DB\$Save\$F:	DB	0	;Flags
00906	044B		DB\$Save\$A:	DB	ō	;A register
00907			DB\$Stack:			;Debug stack area
00908						; The registers in the stack area are PUSHed
00909						; onto the stack and accessed directly.
L						

Figure 10-2. (Continued)

<u> </u>						
00910		;				
00911			er captio	n messao	es.	
00912		;				
00913			bla bala			Display#Cada is used by severe
						Display\$Code is used to access
00914		; the re	gister ca	aption st	ring.	
00915		;				
00916		DB\$Register\$Ca	ptions:			
00917	044C 7204	DW	DB\$F\$RC	2	;Flags	
00918	044E 7804	DW	DB\$A\$RC		;A regi	et ar
00919	0450 7A04	DW	DB\$B\$RC			ster
					; B	
00920	0452 7C04	DW	DB\$C\$RC		;C	
00921	0454 7E04	DW	DB\$D\$RC	2	; D	
00922	0456 8004	DW	DB\$E\$RC	2	;E	
00923	0458 8204	DW	DB\$H\$RC		jΗ	
00924	045A 8404	DW	DB\$L\$RC		j L	
	0450 8604					
00925		DW	DB\$BC\$F		;BC	
00926	045E 8904	DW	DB\$DE\$R		;DE	
00927	0460 8C04	DW	DB\$HL\$R	30	; HL	
00928	0462 8F04	DW	DB\$SP\$R	8C	;Stack	pointer
00929	0464 9204	DW	DB\$M\$RC	:	; Memory	
00930	0466 A604	nw	DB\$B\$BC		; (BC)	
00931	0468 AB04	DW	DB\$B\$DE		;(DE)	
00932	046A B004	DW	DB\$B\$HL		;(HL)	
00933	046C B504	DW	DB\$W\$BC	S\$RC	;(BC+1)	,(BC)
00934	046E C104	DW	DB\$W\$DE	\$RC	;(DE+1)	,(DE)
00935	0470 CD04	DW	DB\$W\$HL		;(HL+1)	
00936						,
00937	0472 4660616773	Decenc-	DB	(E) (0	Flag
				'Flags'	,0	;Flags
00938		DB\$A\$RC:	DB	´A´,O		;A register
00939	047A 4200 1	DB\$B\$RC:	DB	′Β′,Ο		; B
00940	047C 4300 1	DB\$C\$RC:	DB	′C′,O		;C
00941	047E 4400	DB\$D\$RC:	DB	´D´,O		; D
00942			DB	Έ́,ŏ		
		DB\$E\$RC:				;E
00943		DB\$H\$RC:	DB	′H′,O		3H
00944		DB\$L\$RC:	DB	'L',0		۶L
00945	0486 424300 1	DB\$BC\$RC:	DB	′BC′,0		; BC
00946		DB\$DE\$RC:	DB	DE O		; DE
00947		DB\$HL\$RC:	DB	'HL',0		
				HL,U		;HL
00948		DB\$SP\$RC:	DB	'SP',0		;Stack pointer
00949	0492 5374617274		DB	'Start,	End Add	ress 1,0 ;Memory
00950	04A6 28424329001		DB	(BC),	0	;(BC)
00951	04AB 2844452900I	DB\$B\$DE\$RC:	DB	1(DE)1,0 1(HL)1,0	5	;(DE)
00952	04B0 28484C2900I	DR&R&HI &RC.	DB .	- (HE) / (ĥ	; (HL)
00953	04B5 2842432B31I		DB	(DC+1)		
				(BC+1)	, (BC) , 0	;(BC+1),(BC)
00954	04C1 2844452B311		DB	(DE+1)	, (DE)',0	;(DE+1),(DE)
00955	04CD 28484C2B311	DB\$W\$HL\$RC:	DB	<pre>(HL+1);</pre>	,(HL)/,0	;(HL+1),(HL)
00956	1	,				
00957		; Flags	message			
00958						
00959	04D9 43785A784D	, DR#Elage#Mcg.	DB	20070MU		Compatible with DDT's display
00960			00	CALANA		FCOMPACIDIE WICH DDI'S DISPLAY
	;					
00961	;	; Flags	masks use	a to test	t user's	flag byte
00962	;	;				
00963	I	DB\$Flag\$Masks:				
00964	04E4 01		DB	0000\$000	01B	;Carry
00965	04E5 40		DB	0100\$000		Zero
00966	04E6 80		DB	1000\$000		
	0460 00					;Minus
00967	04E7 04		DB	0000\$010		;Even parity
00968	04E8 10		DB	0001\$000	DOB	;Interdigit carry (aux carry)
00969	04E9 00		DB	0		;Terminator
01100	-	; #				
01101			+			
01102						
	1	, inis r	outine in	ILIAI1Zes	s the det	pug package.
01103						
01104	I	DB\$Init:				
01105		IF	DB\$BIOS	\$10		Use BIOS for CONIN/CONOUT
01106		LHLD	1			Get warm boot address from base
01107			-			; page. H = BIOS jump vector page
01108		MVI	L,09H			Page // - Dioo jump vector page
						;Get CONIN offset in jump vector
01109		SHLD		NIN + 1		;Set up address
01110		MVI	L,OCH			;Get CONOUT offset in jump vector
01111		SHLD	BIOS\$CO	NOUT + 1		
01112		ENDIF				
01113						
						tunnakiana ka unanina**
01114						tructions to receive control
01115				; when		instruction is executed
01116	04EA 3EC3	MVI	A, JMP		;Set JMF	P instructions at RST points
1						

01117	04EC 322000	STA	RST4	
01118	04EC 322000 04EF 322800	STA	RST5	
	04EF 322800			- Adduces of false imput washing
01119	04F2 211A08	LXI	H,DB\$Input	Address of fake input routine;
01120	04F5 222100	SHLD	RST4 + 1	
01121	04F8 216C08	LXI	H,DB\$Output	Address of fake output routine
01122	04FB 222900	SHLD	RST5 + 1	
01123	0112 222/00	01.22		
	A.F.F. 00	DET		
01124	04FE C9	RET		
01200		; #		
01201	/	; DB\$CON	INU	
01202		; This r	outine returns t	he next character from the console,
01203				"z" to uppercase letters.
		-	inverting a to	
01204		j		
01205		DB\$CONINU:		
01206	04FF CD0505	CALL	DB\$CONIN	;Get character from keyboard
01207	0502 C31B09	JMP	DB\$A\$To\$Upper	;Fold to upper and return
01300		;#		
01301		; DB\$CON	TN .	
01302				he next character from the console.
01303		; Accord	ing to the setti	ng of equates, it uses simple
01304		; polled	I/O, the BDOS (function 2) or the BIOS.
01305		;		
01306		; Exit p	arameters	
01307		;		
01308		,	A = character	from console
		,	H = character	TTOM CONSOLE
01309		1		
01310		DB\$CONIN:		
01311		IF	DB\$Polled\$IO	Simple polled input
01312		IN	DB\$Status\$Port	
01313		ANI	DB\$Input\$Ready	
		JZ	DB\$CONIN	:No
01314				
01315		IN	DB\$Data\$Port	;Input data character
01316		PUSH	PSW	;Save data character
01317		MOV	C,A	;Ready for output
01318		CALL	DB\$CONOUT	;Echo it back
01319		POP	PSW	Recover data character
01320		RET	1.044	frecover data character
01321		ENDIF		
01322				
01323		IF	DB\$BDOS\$IO	;Use BDOS for input
01324	0505 0E01	MVI	C.B\$CONIN	Read console
01325	0507 C30500	JMP	BDOS	;BDOS returns to our caller
	0307 030300		6003	Jobos returns to our carren
01326		ENDIF		
01327				
01328		IF	DB\$BIOS\$IO	;Use BIOS for input
01329		JMP	BIOS\$CONIN	This was set up during BIOS
01330				; initialization
		' ENDIF		, initialization
01331		ENDIF		
01332				
01400		; #		
01401		; DB\$CON	OUT	
01402			outine outputs t	he character in the C register to the
01403				polled I/O, the BDOS or the BIOS.
			e, darud grubi6	Forres 1/0, the bbos of the bros.
01404		,		
01405		; Entry	parameters	
01406		;	A = byte to be	output
01407		;		
01408		DB\$CONOUT:		
01409	050A 3A0604	LDA	DB\$Flag	;Check if debug output enabled
				touers to georg output endoired
01410	050D B7	ORA	A	· *
01411	050E C8	RZ		;Ignore output if disabled
01412		-		
01413		IF	DB\$Polled\$IO	;Use simple polled output
01414		IN	DB\$Status\$Port	Check if ready for output
01415		ANI	DB\$Output\$Read	
01416		JZ	DB\$CONOUT	; No
01417		MOV	A,C	;Get data byte
01418		OUT	DB\$Data\$Port	
01419		RET		
01420		ENDIF		
01421				
		IF	DB\$BDOS\$IO	;Use BDOS for output
01422				
01423	050F 59	MOV	E,C	;Move into correct register
01424	0510 OE02	MVI	C,B\$CONOUT	
01425	0512 C30500	JMP	BDOS	;BDOS returns to our caller
01426		ENDIF		• • • • • • • • • • • • • • • • • • • •
		ENDIF		
01427			DD4D700110	Was BIOC for autout
01428		IF ,	DB\$BIOS\$IO	;Use BIOS for output
L				

Figure 10-2. (Continued)

429 430 431		MC JL EN)V A,C 1P BIOS\$ NDIF	CONOUT	;Move into correct register ;Set up during debug initializatio
500		;#			
501		1			
502		,	3\$On		
503					ebug output by setting the
504		; DE	3\$Flag nonze	ro.	
505 506		; DB\$On:			
506	0515 F5		JSH PSW		Processe senisters
508	0516 3EFF	M		-u	;Preserve registers
509	0518 320604	SI		 .ao	;Set control flag on
510	051B F1	PC	P PSW		, or control (ing on
511	0510 09	RE			
600		;#			
601		;			
602		; DE	B\$Off		
603		; TH	is routine	disables all	debug output by setting the
604		; DE	3\$Flag to ze	ro.	
605 606					
605 607	051D F5	DB\$Off:	JSH PSW		· Prosente mani-t
608	051E AF	XF			;Preserve registers
609	051F 320604	S1		a0	;Clear control flag
610	0522 F1	PC		~	yorean control fidy
611	0523 C9	RE			
700		;#			
701		;			
702		; DE	3\$Set\$Pass		
703					counter. Subsequent calls to DB\$Pass
704				count, and w	hen it reaches 0, debug output
705		-	enabled.		
706		;			
707			lling seque	nce	
708 709		1	CAL 1		_
710		;	CALL DW	DB\$Set\$Pas Pass\$Count	
711		,	DW	rasspucint	**8102
712		, DB\$Set\$Pas	s:		
713	0524 220904		ILD DB\$Sa	ve\$HL	;Preserve user's HL
714	0527 E1	PO			Recover return address
715	0528 D5		ISH D		Preserve user's DE
716	0529 5E	MO			;Get LS byte of count
717	052A 23	IN			;Update pointer
718	052B 56	MO			;Get MS byte
719	0520 23	IN			;HL points to return address
720	052D EB		HG		;HL = pass counter
721	052E 220704			ss\$Count	;Set debug pass counter
722 723	0531 EB 0532 D1	XC P0	HG PD		;HL points to return address
724 724	0533 E3		ir D HL		;Recover user's DE ;Recover user's HL and set
725	3000 23				; return address on top of stack
726	0534 C9	RE	т		, return address on top Of Stack
300		;#			
301		;			
302			\$Pass		
303			is routine	decrements th	e debug pass counter -
304					it takes no further action.
305					sets the debug control flag nonzero
306			enable-deb		
307		;			
308		DB\$Pass:			
309	0535 F5	PU			;Save user's registers
310	0536 E5	PU			
811	0537 2A0704	LH		ss\$Count	;Get pass count
312 313	053A 2B	DC			Charle if sound and have been block
313 314	0538 7C 053C 87	MO			;Check if count now negative
314 315	0530 FA4705	JM		c = \$ v	Vec take no finithen solio-
	0540 220704	SH		ss≆x ss\$Count	;Yes, take no further action ;Save downdated count
316	0543 B5	OR		ssecount	;Save downdated count ;Check if count now zero
		JZ		ss\$FD	;Yes, enable debug
B17	0544 CA4405		DD#Fa:	33760	, ies, endure debug
B17 B18	0544 CA4A05				•
B17 B18 B19		DB\$Pass\$x:	р н		; :Recover user's registers
816 817 818 819 820 821	0 544 CA4A05 0547 E1 0548 F1				; ;Recover user's registers

01823		;			
01824		DB\$Pass\$Ed:			;Enable debug
01825 01826	054A 3EFF 054C 320604	MVI STA	A,OFFH DB\$Flag		;Set debug control flag
01827	054F C34705	JMP	DB\$Pass\$	v	, set bebug control ring
01900	0041 004700	;#	2247 4337	^	
01901		7			
01902		; DB\$Disp			
01903	1		the prim	ary debug disp	play routine.
01904 01905		; : Calling	sequence		
01906		;	Jequence		
01907		;		DB\$Display	
01908		;		Display\$Code	
01909		;	DB	'Caption Strin	יg′,0
01910 01911		;	Dicolar	oode identifi	es which register(s) are to be
01912			displaye		es which register(s) are to be
01913		;			
01914		;			specifies a block of memory
01915		7 ,	the sequ	ence is:	
01916		7	CALL	DB\$Display	
01917 01918			DB	DB\$D15Play Display\$Code	
01919			DW	Start\$Address,	,End\$Address
01920		;	DB	'Caption Strin	
01921		;			
01922		DB\$Display:			
01923 01924		; DB\$Display\$Enab	led•		
01925	0552 220904	SHLD	DB\$Save\$	H	;Save user's HL
01926					,
01927	0555 E3	X THL			;Get return address from stack
01928	0556 220D04	SHLD	DB\$Save\$	RA	;This gets updated by debug code
01929	0559 E5	PUSH	н		;Save return address temporarily
01930 01931	055A 2B 055B 2B	DCX DCX	H H		;Subtract 3 to address call instruction ; itself
01931	0556 2B	DCX	н		; ((Sel)
01933	055D 220F04	SHLD	DB\$Call\$	Address	Save actual address of CALL
01934	0560 E1	POP	н		Recover return address
01935					
01936	0561 F5	PUSH	PSW		;Temporarily save flags to avoid ; them being changed by DAD SP
01938	0562 210000	LXI	н, о		Preserve stack pointer
01939	0565 39	DAD	SP		fileselve stack pointer
01940	0566 23	INX	н		;Correct for extra PUSH PSW needed
01941	0567 23	INX	н		; to save the flags
01942	0568 220B04	SHLD	DB\$Save\$	SP	;Recover flags
01943 01944	056B F1	POP	PSW		;Recover flags
01945	056C 314C04	LXI	SP,DB\$St	ack	;Switch to local stack
01946			,		
01947	056F F5	PUSH	PSW		;Save other user's registers
01948	0570 C5	PUSH	в		;The stack area is specially laid
01949	0571 D5	PUSH	D		; out to access these registers
01950 01951	0572 2A0D04	LHLD	DB\$Save\$	RA	;Get return address
01951	0575 7E	MOV	A,M		;Get display code
01953	0576 321504	STA	DB\$Displ	ay\$Code	
01954	0579 23	INX	н		;Update return address
01955			DD+//		
01956	057A FE18	CPI	DB\$M DB\$Not\$N		;Check if memory to be displayed
01957 01958	057C C29105 057F 5E	JNZ MOV	DB\$Not\$P E,M	nemory	;Get DE = start address
01958	0580 23	INX	H		you be - start address
01960	0581 56	MOV	D, M		
01961	0582 23	INX	н		
01962	0583 EB	XCHG			;HL = start address
01963	0584 221104	SHLD	DB\$Start	\$Address	;HL -> end address
01964	0587 EB 0588 5E	XCHG MOV	E,M		;AL -> end address ;Get DE = end address
01965	0589 23	INX	H		your and all and addred a
01967	058A 56	MOV	D, M		
01968	058B 23	INX	н		
01969	058C EB	XCHG			;HL = end address, DE -> caption
01970	058D 221304	SHLD	DB\$End\$4	Address	;HL -> caption string
01971	0590 EB	XCHG			AUT > CENTROL STITUE

Figure 10-2. (Continued)

01972		DB\$Not\$Memory:		
01973		*		
01974 01975			preamble and caption st	
01976		; The fo	rmat for everything excep	ot memory display is:
01977			Caption String : $RC = v_{1}$	
01978				
01979		; Call A	ddress ¦ Va	alue
01980		f.		ion (A, B, C)
01981		,		
01982				s output at the start of the
01983		; messag	e - but NOT at the end.	
01984		7		
01985			displays look like :	
01986		,		
01987 01988				hh hh hh hh hh : cece cece cece
01989		, 5555 1 101 100		in in mini in in in i cece cece cece
01990		,		
01991	0591 E5	PUSH	н	Save pointer to caption string
01992	0592 CDC107	CALL	DB\$CRLF	;Display carriage return, line feed
01993	0595 CD7C07	CALL	DB\$Display\$CALLA	Display DB\$Call\$Address in hex.
01994				
01995	0598 E1	POP	н	Recover pointer to caption string
01996		DB\$Display\$Cap		;HL -> caption string
01997	0599 7E	MOV	A, M	;Get character
01998	059A 23	INX	н	
01999	059B B7	ORA	A	;Check if end of string
02000 02001	059C CAA805	JZ	DB\$End\$Caption	;Yes
02002	059F E5	PUSH	н	;Save string pointer
02003	05A0 4F	MOV	С, A	Ready for output
02004	05A1 CD0A05	CALL	DB\$CONOUT	;Display character
02005	05A4 E1	POP	H	Recover string pointer
02006	05A5 C39905	JMP	DB\$Display\$Caption	;Go back for next character
02007				
02008		DB\$End\$Caption		
02009	05A8 220D04	SHLD	DB\$Save\$RA	;Save updated return address
02010				
02011	05AB CDC807	CALL	DB\$Colon	;Display 1 : 1
02012				
02013				Display register caption
02014	05AE 3A1504	LDA	DB\$Display\$Code	;Get user's display code
02015	05B1 5F	MOV	E,A	;Make display code into word
02016	05B2 1600	MVI	D, O	
02017	05B4 D5	PUSH	D	;Save word value for later
02018 02019	05B5 FE18	681	PD+M	
02020	0585 FE18 0587 CACF05	CPI JZ	DB\$M DB\$PiceloutMontContine	Memory display is a special case
02021	0367 CHOP00	52	DB\$Display\$Mem\$Caption	;Yes
02022	05BA 214C04	LXI	H,DB\$Register\$Captions	;Make pointer to address in table
02023	05BD 19	DAD	D	;HL -> word containing address of
02024			-	; register caption
02025	05BE 5E	MOV	E,M	;Get LS byte of address
02026	05BF 23	INX	н	
02027	0500 56	MOV	D, M	;DE -> register caption string
02028	05C1 EB	XCHG		;HL -> register caption string
02029	05C2 CDEE07	CALL	DB\$MSG	;Display message addressed by HL
02030		-		
02031	05C5 CDD607 05C8 203D2000	CALL	DB\$MSGI	;Display in-line message
02032	05C8 203D2000 05CC C3ED05	DB JMP	4 = 4,0	
02033	03CC C3ED05	JMP .	DB\$Select\$Routine	;Go to correct processor
02034		; DB\$Display\$Mem	*Cantion:	The memory display yequives a
02036		PPADISHIG AUGU	roaptioni	The memory display requires a special ; caption with the start and end
02037				; caption with the start and end; ; addresses
02038	05CF 219204	LXI	H, DB\$M\$RC	; addresses ;Display specific caption
02039	05D2 CDEE07	CALL	DB\$MSG	
02040	05D5 CDC807	CALL	DB\$Colon	;Display ': '
02041				
02042	05D8 2A1104	LHLD	DB\$Start\$Address	;Display start address
02043	05DB CD8707	CALL	DB\$DHLH	;Display HL in hex.
02044				
02045	05DE CDD607	CALL	DB\$MSGI	;Display in-line message
02046	05E1 2C2000	DB	1, 1,0	
02047	0EF4 011004			
02048	05E4 2A1304	LHLD	DB\$End\$Address	;Get end address
L				

02049 02050	05E7 CD8707 05EA CDC107	CAL		;Display HL in hex. ;Display carriage return, line feed
02051				;Drop into select routine
02052		DB\$Select\$		
02053	05ED D1	POF		Recover word value Display\$Code;
02054	05EE 210A06	LX		
02055	05F1 19	DAI	D D	;HL -> address of code to process
02056				; display requirements
02057	05F2 5E	MON	/ E,M	;Get LS byte of address
02058	05F3 23	IN	(н	;Update pointer
02059	05F4 56	MO		;Get MS byte of address
02060	05F5 EB,	XCI		;HL -> code
02061	0010 20,	XCI	10	, iic / cobe
				·Faka liak an stack
02062	05F6 11FB05	LX		;Fake link on stack
02063	05F9 D5	PUS		
02064	05FA E9	PCI	4L ,	;"CALL" display processor
02065		;		
02066		DB\$Exit:		;Return to the user
02067	05FB D1	POF	, D	;Recover user's registers saved
02068	05FC C1	POF	, в	; on local debug stack
02069	05FD F1	POF		• –
02070	05FE 2A0B04	LHL		;Revert to user's stack
	00FE 2R0804	SPI		Thever to does a stack
02071	0601 F9			· Det undeted voture address / humans
02072	0602 2A0D04	LHL	.D DB\$Save\$RA	;Get updated return address (bypasse
02073				; in-line parameters)
02074	0605 E3	XTI		Replace on top of user's stack;
02075	0606 2A0904	LHL		;Get user's HL
02076	0609 C9	RE	r ,	;Transfer to correct return address
02077			-	
02078				
02079		DB\$Display	Table:	
	0400 3004	DD#DIspiay. DW	DP\$F	;Flags
02080	060A 3006		DP\$A	
02081	060C 5406	DW		;A register
02082	060E 5A06	DW	DP\$B	;B
02083	060E 5A06 0610 6006	DW	DP\$C	;C
02084	0612 6606	DW	DP\$D	; D
02085	0614 6006	DW	DP\$E	;E
02086	0616 7206	DW	DP\$H	2 H
02087	0618 7806	DW	DP\$L	;L
02088		DW	DP\$BC	, E , BC
	061A 7E06			; BC
02089	061C 8406 061E 8A06	DW	DP\$DE	;DE
02090	061E 8A06	DW	DP\$HL	;HL
02091	0620 9006	DW	DP\$SP	;Stack pointer
02092	0622 9606	DW	DP\$M	;Memory
02093	0624 4907	DW	DP\$B\$BC	;(BC)
02094	0626 5007	DW	DP\$B\$DE	;(DE)
02095	0628 5707	DW	DP\$B\$HL	;(HL)
02096	062A 5E07	DW	DP\$W\$BC	; (BC+1), (BC)
02097	062C 6807	DW	DP\$W\$DE	;(DE+1),(DE)
	0620 8807	DW	DP\$W\$HL	
02098	062E 7207			;(HL+1),(HL)
02200		;#		
02201		; Del	oug display proc	ssing routines
02202		;		
02203		DP\$F:		;Flags
02204				;The flags are displayed in the same way the
02205				; DDT uses: C1ZOMOEOIO
02206	0630 3A4A04	LD	A DB\$Save\$F	:Get flags
02207	0633 47	MO		Preserve COPY
02208	0634 21DA04	LX		Msg + 1 ;HL -> first 0/1 in message
				$\frac{1}{2} + \frac{1}{2} + \frac{1}$
02209	0637 11E404		. <i>D</i> , DD⊅r1ag\$	1asks ;DE -> table of flag mask values
02210		DB\$F\$Next:		
02211	063A 1A	LD		;Get next flag mask
02212	063B B7	OR		;Check if end of table
02213	063C CA4E06	JZ	DB\$F\$Displ	ay ;Yes, display the results
02214				
02215	063F A0	AN	A B	;Check if this flag is set
02216	0640 3E31	MV		Assume yes
		JN		
02217	0642 C24706			;Yes, it is set
02218	0645 3E30	MV	I A, 101	;No,it is clear
02219		DB\$F\$NZ:		
02220	0647 77	MO		Store '0' or '1' in message text
02221	0648 23	IN		;Update pointer to next 0/1
02222	0649 23	IN	хн	
02223	064A 13	IN		;Update flag mask pointer
				· · · · · · · · · · · · · · · · · · ·
02224	064B C33A06	JM		
02225		DB\$F\$Displ		;Display results
02226	064E 21D904	LX		

Figure 10-2. (Continued)

02227	0451 00000		IMD	DRANCO	
02227 02228	0651 C3EEC		JMP	DB\$MSG	;Display message and return
		7 DD#A			
02229	0/F4 04-07	DP\$A:	1 54	;A register	
02230	0654 3A4B0 0657 C3910		LDA	DB\$Save\$A	;Get saved value
02231	0637 63910	<i></i>	JMP	DB\$DAH	;Display it and return
02232		; DD#D			
02233	0/EA 04 00	DP\$B:	1.54	; B	
02234	065A 3A490 065D C3910		LDA	DB\$Save\$B	;Get saved value
02235	0650 03910)/	JMP	DB\$DAH	;Display it and return
02236		; DP\$C:		. •	
02237				;C	
02238	0660 3A480 0663 C3910		LDA JMP	DB\$Save\$C DB\$DAH	;Get saved value
	0663 03910	,,	JMP	DB#DAH	Display it and return
02240		;		_	
02241	A	DP\$D:		; D	
02242	0666 3A470		LDA	DB\$Save\$D	;Get saved value
02243	0669 C3910	,,	JMP	DB\$DAH	;Display it and return
02244		; DP\$E:		÷E	
	066C 3A460		1.04		• Call actual training
02246	066F C3910	24 27		DB\$Save\$E	;Get saved value
02247	U00F 03910	·/ .	JMP	DB\$DAH	;Display it and return
		; DP\$H:		:н	
02249	0470 04044		1.554		Cot and the last
02250	0672 3A0A0 0675 C3910			DB\$Save\$H	;Get saved value
02251	06/3 03910	·· .	JMP	DB\$DAH	;Display it and return
02252		;			
02253	0470 04000	DP\$L:	1.54	;L DB#Countl	Cot could up hus
02254 02255	0678 3A090 0678 C3910		LDA JMP	DB\$Save\$L DB\$DAH	;Get saved value
02255	VO/B 1.3710	·· .	One.	DDADWU	Display it and return
02256		; DP\$BC:		: BC	
02257	067E 2A480		LHLD	;BC DB\$Save\$C	.Get caved word value
02258	0681 03870		JMP	DB\$SAVE\$C DB\$DHLH	;Get saved word value ;Display it and return
02259	0001 038/0		one	DDFUNLN	jutapidy it and return
02260		; DP\$DE:		; DE	
02262	0684 24460		LHLD	DB\$Save\$E	;Get saved word value
02262	0687 C3870		JMP	DB\$DHLH	;Display it and return
02263	0007 00870	•	0.11	DD#DHLH	, propras It and return
02265		; DP\$HL:		:HL	
02265	068A 2A090		LHLD	;nc DB\$Save\$HL	;Get saved word value
02267	068D C3870		JMP	DB\$DHLH	;Display it and return
02268	0000 038/0	•	011	55#DACA	Josephaj It and return
02268		, DP\$SP:		Stack Pointer	
02270	0690 2A0B0		LHLD	DB\$Save\$SP	;Get saved word value
02271	0693 C3870		JMP	DB\$Savessr DB\$DHLH	;Display it and return
02272	0070 00070	•	0.1	DD#DHLH	, prepras It and return
02273		DP\$M:		:Memory	
02274	0696 2A130		LHLD	DB\$End\$Address	;Increment end address to make
02275	0699 23	•	INX	H	; arithmetic easier
02276	069A 22130	4	SHLD	DB\$End\$Address	,
02277					
02278	069D 2A110)4	LHLD	DB\$Start\$Addres	5
02279	06A0 CD3A0	07	CALL	DB\$M\$Check\$End	;Compare HL to End\$Address
02280	06A3 DAD10		JC	DB\$M\$Address\$OK	:End > start
02281	06A6 CDD60		CALL	DB\$MSGI	;Error start > end
02282	06A9 0D0A2		DB		DR - Start Address > End **1,0
02283	06CD C9		RET	Serry Villy An ERRU	AN OVER CHEMICES / LINE AN IV
02284					
02285		, DB\$M\$Ne	xt\$Line:		
02286	06CE CDC10		CALL	DB\$CRLF	;Output carriage return, line feed
02287			dress\$0K		;Bypass CR,LF for first line
02288	06D1 CDD60		CALL	DB\$MSGI	;Indent line
02289	06D4 20200		DB	,0	
02290	06D7 2A110		LHLD	DB\$Start\$Address	;Get start of line address
02291	06DA CD870		CALL	DB\$DHLH	;Display in hex
02292					, , ,
02293	06DD CDC80	7	CALL	DB\$Colon	;Display ′; ′
02294					
02295	06E0 2A110	4	LHLD	DB\$Start\$Address	5
02296			x t\$Hex\$ B		
02297	06E3 E5		PUSH	н	;Save memory address
02298	06E4 CDD00	7	CALL	DB\$Blank	;Output a blank
02299	06E7 E1		POP	н	Recover current byte address
02300	06E8 7E		MOV	Α,Μ	;Get byte from memory
02301	06E9 23		INX	н	;Update memory pointer
02302	06EA E5	_	PUSH	н	;Save for later
02303	06EB CD910	7	CALL	DB\$DAH	;Display in hex.
02304	06EE E1		POP	н	Recover memory updated address;
					

Figure 10-2. (Continued)

00005	0/55 000403			Company III we and address
02305	OGEF CD3A07	CALL	DB\$M\$Check\$End DB\$M\$Display\$AS	;Compare HL vs.end address CII ;Yes, end of area
02306 02307	06F2 CAFE06	MOV	A.L	Check if at start of new line,
02307	06F5 7D	ANI	0000\$1111B	; (is address XXX0H?)
	06F6 E60F		DB\$M\$Display\$AS	
02309	06F8 CAFE06 06FB C3E306	JZ JMP	DB\$M\$Next\$Hex\$B	
02310	06FB C3E306	UMP	DB#H#Nex(#nex#b	yte ino, toop back for another
02311		; 		Discharge is ACCII
02312		DB\$M\$Display\$A		;Display bytes in ASCII ;Display ´ : ´
02313	06FE CDC807	CALL	DB\$Colon	
02314	0701 2A1104	LHLD	DB\$Start\$Addres	s ;Start ASCII as beginning of line
02315		DB\$M\$Next\$ASCI		
02316	0704 7E	MOV	A, M	;Get byte from memory
02317	0705 E5	PUSH	н	Save memory address
02318	0706 E67F	ANI	0111\$1111B	;Remove parity
02319	0708 4 F	MOV	C,A	Prepare for output
02320	0709 FE20	CPI		;Check if non-graphic
02321	070B D21007	JNC	DB\$M\$Display\$Ch	
02322	070E 0E2E	MVI	C,′.′	;Display non-graphic as '.'
02323		DB\$M\$Display\$C		
02324	0710 FE7F	CPI	7FH	<pre>;Check if DEL (may be non-graphic)</pre>
02325	0712 C21707	JNZ	DB\$M\$Not\$DEL	;No, it is graphic
02326	0715 0E2E	MVI	C, 1.1	;Force to 1.1
02327		;		
02328		DB\$M\$Not\$DEL:		
02329	0717 CD0A05	CALL	DB\$CONOUT	;Display character
02330	071A E1	POP	н	Recover memory address
02331	071B 23	INX	Н	;Update memory pointer
02332	0710 221104	SHLD	DB\$Start\$Addres	
02333	071F CD3A07	CALL	DB\$M\$Check\$End	;Check if end of memory dump
02334	0722 CA3707	JZ	DB\$M\$Exit	;Yes, done
02335	0725 7D	MOV	A,L	;Check if end of line
02336	0726 E60F	ANI	0000\$1111B	<pre>; by checking address = XXXOH</pre>
02337	0728 CACE06	JZ	DB\$M\$Next\$Line	;Yes, start next line
02338	072B 7D	MOV	A,L	;Check if extra blank needed
02339	072C E603	ANI	0000\$0011B	; if address is multiple of 4
02340	072E C20407	JNZ	DB\$M\$Next\$ASCII	Byte ;No go back for next character
02341	0731 CDD007	CALL	DB\$Blank	;Yes, output blank
02342	0734 C30407	JMP	DB\$M\$Next\$ASCII	
02343				
02344				
02345		DB\$M\$Exit:		
02346	0737 C3C107	JMP	DB\$CRLF	;Output carriage return, line feed
02347				; and return
02348				,
02349		DB\$M\$Check\$End		;Compares HL vs End\$Address
02350	073A D5	PUSH	ם	;Save DE (defensive programming)
02351	073B EB	XCHG	-	;DE = current address
02352	073C 2A1304	LHLD	DB\$End\$Address	;Get end address
02353	073F 7A	MOV	A.D	;Compare MS bytes
02354	0740 BC	CMP	H C	,
02355	0741 C24607	JNZ	DB\$M\$Check\$End\$	X ;Exit now as they are unequal
02356	0744 7B	MOV	A,E	Compare LS bytes
02357	0745 BD	CMP	1	,
02358	07.10 22	DB\$M\$Check\$End	\$X.	
02359	0746 EB	XCHG		;HL = current address
02360	0747 D1	POP	D	Recover DE
02361	0748 C9	RET	-	Return with condition flags set
02362	07.10 07	1		yneten nath tenetten raugs set
02363		DP\$B\$BC:	;(BC)	
02363	0749 284804	LHLD	DB\$Save\$C	;Get saved word value
02365	074C 7E	MOV	A,M	;Get byte addressed by it
02366	074D C39107	JMP	DB\$DAH	;Display it and return
02368	0/40 00/10/		224000	yearay at did letdin
02368		, DP\$B\$DE:	;(DE)	
02369	0750 2A4604	LHLD	DB\$Save\$E	;Get saved word value
02370	0753 7E	MOV	A,M	;Get byte addressed by it
02370	0754 C39107	JMP	DB\$DAH	;Display it and return
02372	0/04 03/10/		2070111	parameters as which resource
02372		, DP\$B\$HL:	;(HL)	
02373	0757 2A0904	LHLD	DB\$Save\$HL	;Get saved word value
02374	075A 7E	MOV	A,M	Get byte addressed by it
02375	075B C39107	JMP	DB\$DAH	;Display it and return
02376	0/36 03710/	•	DD#DAN	, present at and retain
		; DP\$W\$BC:	;(BC+1),(BC)	
02378	075E 2A4804	LHLD	DB\$Save\$C	;Get saved word value
02379	075E 2A4804 0761 5E	MOV	E,M	;Get word addressed by it
	0762 23	INX	E, m H	YOEL WOLD BUDDED DY IL
02381	V/02 23	TINA		

Figure 10-2. (Continued)

02382	0763	56		MOV	D,M	
02383	0764			XCHG		;HL = word to be displayed
02384		C38707		JMP	DB\$DHLH	Display it and return
02385			;			
02386			DP\$W\$DE:	•	;(DE+1),(DE)	
02387	0768	2A4604		LHLD	DB\$Save\$E	;Get saved word value
02388	076B	5E		MOV	E,M	;Get word addressed by it
02389	076C	23		INX	н	,
02390	076D			MOV	D. M	
02391	076E	EB		XCHG	-	;HL = word to be displayed
02392		C38707		JMP	DB\$DHLH	Display it and return
02393			:			
02394			DP\$W\$HL:		;(HL+1),(HL)	
02395	0772	2A0904		LHLD	DB\$Save\$HL	;Get saved word value
02396	0775			MOV	E,M	;Get word addressed by it
02397	0776			INX	н	
02398	0777	56		MOV	D, M	
02399	0778	EB		XCHG		;HL = word to be displayed
02400	0779	C38707		JMP	DB\$DHLH	Display it and return
02401			;			
02500			; #			
02501			;	DB\$Disp	lay\$CALLA	
02502			;			he DB\$Call\$Address in hexadecimal,
02503			;		d by " : ".	
02504			:			
02505			DB\$Disp	lay\$CALL	A:	
02506	077C	E5		PUSH	н	;Save caller's HL
02507		2A0F04		LHLD		;Get the call address
02508		CD8707		CALL	DB\$DHLH	;Display HL in hex.
02509	0783			POP	H	Recover caller's HL
02510	0784	C3C807		JMP	DB\$Colon	;Display " : " and return
02511			,			
02600			;#			
02601			;			
02602			;	DB\$DHLH		
02603			;	Display	HL in hex.	
02604			;			
02605			;	Entry p	arameters	
02606			;			
02607			;		HL = value to be	e displayed
02608			;			
02609			DB\$DHLH:			
02610	0787			PUSH	н	;Save input value
02611	0788			MOV	A,H	;Get MS byte first
02612	0789	CD9107		CALL	DB\$DAH	;Display A in hex.
02613	078C			POP	н	Recover input value
02614	078D	70		MOV	A,L	;Get LS byte
02615	078E	C39107		JMP	DB\$DAH	;Display it and return
02616			;			
02700			;#			
02701			;			
02702				DB\$DAH		
02703			;	Display	A register in he	xadecimal
02704			;			
02705				Entry p	arameters	
02706			;			
02707			;		A = value to be	converted and output
02708			;			
02709			DB\$DAH:			
02710	0791			PUSH	PSW	<pre>#Take a copy of the value to be converted</pre>
02711	0792			RRC		;Shift A right four places
02712	0793			RRC		
02713	0794			RRC		
02714	0795			RRC		
02715		CDB407		CALL	DB\$Nibble\$To\$He>	
02716		CDOA05		CALL	DB\$CONOUT	;Display the character
02717	079C			POP	PSW	;Get original value again
02718		CDB407		CALL	DB\$Nibble\$To\$He>	
02719	07A0 (C30A05		JMP	DB\$CONOUT	Display and return to caller;
02800			;#			
02801			;			
02802				DB\$CAH		
02803						exadecimal ASCII and store in
02804				specifie	ed address.	
02805			;			
02806				Entry pa	arameters	
02807			;			
1						
1						

Г						
	02808		;		A = value to be	converted and output
	02809		;		HL -> buffer ar	ea to receive two characters of output
	02810		;			
	02811		;	Exit pa	rameters	
	02812		;			
L	02813		;		HL -> byte foll	owing last hex.byte output
	02814		;			
L	02815		DB\$CAH:			
L	02816	07A3 F5		PUSH	PSW	;Take a copy of the value to be converted
1	02817	07A4 OF		RRC		;Shift A right four places
	02818	07A5 OF		RRC		
ì	02819	07A6 OF		RRC		
L	02820	07A7 OF		RRC		0
	02821	07A8 CDB407		CALL	DB\$Nibble\$To\$He	
	02822	07AB 77		MOV	M,A H	;Save in memory ;Update pointer
L	02823	07AC 23		INX POP	PSW	
	02824	07AD F1				;Get original value again
L	02825	07AE CDB407 07B1 77		CALL	DB\$Nibble\$To\$He	
	02826	07B2 23		MOV INX	M,A H	;Save in memory ;Update pointer
1	02827 02828	07B3 C9		RET	•	jopuate pointer
L		0783 09	- 44	REI		
1	02900		;#			
1	02901 02902		:	Miner -	ubroutines	
1	02902		7	minor s	doi out mes	
1	02903		:			
1	02904		; ;	RENISH	lesTosHex	
L	02906		;			ine that converts the least
Ł	02907		;			f the A register into an ASCII
	02908		;		aracter in A and	
	02909		;	11641 CI	anacter in A and	•
	02910		;	Entry p	arameters	
L	02911		;			
	02912		;		A = nibble to b	e converted in LS 4 bits
L	02913		;			
L	02914		;	Exit pa	rameters	
L	02915		;			
L	02916		;		A,C = ASCII he	x. character
	02917		;			
L	02918		DB\$Nibb	le\$To\$He	×:	
	02919	07B4 E60F		ANI	0000\$1111B	;Isolate LS four bits
L	02920	07B6 C630		ADI	'0' '9' + 1	Convert to ASCII
1	02921	0788 FE3A		CPI		;Compare to maximum
T	02922	07BA DABF07		JC		;No need to convert to A -> F
T	02923	07BD C607		ADI		;Convert to a letter
1	02924			Numeric:		
	02925	07BF 4F		MOV	C,A	For convenience of other routines
	02926	07C0 C9		RET		
	02927					
	02928				1. N. 19	
	02929		;			
1	02930		2	DB\$CRLF	voution to dic-1	ay carriage return, line feed.
L	02931		,	SIMPIC	routine to uispi	ay cantage (erain, true teen.
1	02932		; DB\$CRLF			
1	02933	A701 000/00	DDPCNLF		DRANCOT	Disalay in line property
	02934	07C1 CDD607 07C4 0D0A00		CALL		;Display in-line message
T	02935	0704 UDUAUU		DB RET	ODH, OAH, O	
	02936	07C7 C9				
1	02937 02938		:	DB\$Colo	n	
	02938		:			av / • /
L	02939		!	21mb16	routine to displ	
1	02940		; DB\$Colo			
	02942	07C8 CDD607		CALL	DB\$MSGI	;Display in-line message
T	02943	07CB 203A2000		DB	1: 1,0	for sprant and a sine mersing a
T	02944	07CF C9		RET	• • •	
I	02945		;			
	02946		;	DB\$Blan	k	
	02947		;		routine to displ	ay ('.
1	02948		;			
	02949		DB\$Blan	k:		
	02950	07D0 CDD607		CALL	DB\$MSGI	;Display in-line message
	02951	07D3 2000		DB	1,0	
	02952	07D5 C9		RET		
	03100		; #			
	03101		;			
I	03102		;	Message	processing subr	outines
L						

Figure 10-2. (Continued)

r						
03103		;				
03104		;	DRAMSOT	(messag	e in-line	a)
			0040301	Chiessay	e 10-110	
03105		Ŧ				ated message that follows the
03106		;	CALL to	MSGOUTI		
03107						
03108		;	Callins	sequenc	-	
			Carring	sequenc	e	
03109		;				
03110		;		CALL	DB\$MSGI	
03111		;		DB	Message	e' 0
03112		7		nex	t instru	CTION
03113		;				
03114		:	Exit pa	rameters		
03115		;				on following message
03116						on fortowing message
		;				
03117		;				
03118		DB\$MSGI	:			
03119						;Get return address of stack, save
03120						
						; user's HL on top of stack
03121	07D6 E3		XTHL			;HL -> message
03122						
03123	07D7 F5		PUSH	PSW		;Save all user's registers
03124	07D8 C5		PUSH	в		youre are aver a registers
03125	07D9 D5		PUSH	D		
03126		DB\$MSGI	\$Next:			
03127	07DA 7E		MOV	A,M		;Get next data byte
	07DB 23					
03128			INX	н		;Update message pointer
03129	07DC B7		ORA	Α		;Check if null byte
03130	07DD C2E507		JNZ	DB\$MSGI	С	;No, continue
03131						
	0750 51		POP	п		· Deservery warmade warmint
03132	07E0 D1			~		;Recover user's registers
03133	07E1 C1		POP	в		
03134	07E2 F1		POP	PSW		
03135	07E3 E3		XTHL			Recover user's HL from stack, replacing
	0/23 23		ATHE			
03136						; it with updated return address
03137	07E4 C9		RET			;Return to address after 00-byte
03138						; after in-line message
		DDAMOOT.	^ -			, ditti in iine message
03139		DB\$MSG I				
03140	07E5 E5		PUSH	н		;Save message pointer
03141	07E6 4F		MOV	C,A		;Ready for output
03142	07E7 CD0A05		CALL	DB\$CONO	нт	,,
					01	. P
03143	07EA E1		POP	н		;Recover message pointer
03144	07EB C3DA07		JMP	DB\$MSGI	\$Next	;Go back for next char.
03145						
03146		-				
		;				
03147		;	DB\$MSG			
03148		;	Output	null-byt	e termin	ated message
03149		;				
			o			
03150		;	Calling	sequenc	e	
03151		;				
03152		;	MESSAGE	:	DB	'Message',0
03153		;		•.		2
					U MECON	ee
03154		;		LXI	H, MESSA	UC
03155		;		CALL	DB\$MSG	
03156		;				
03157			Evit	rameters		
		1	CALC Pa			A
03158		;		HL -> n	uli byte	terminator
03159		;				
03160		:				
03161		DB\$MSG:				
		DD#1130:	-			
03162	07EE F5		PUSH	PSW		;Save user's registers
03163			PUSH	в		
03164	07EF C5		PUSH	ñ		
,						
021/5	07EF C5 07F0 D5	DRAMOOA		D		
03165	07F0 D5	DB\$MSG\$	Next:	-		
03166	07F0 D5 07F1 7E	DB\$MSG\$	Next: MOV	- A, M		;Get next byte for output
	07F0 D5	DB\$MSG\$	Next:	-		;Get next byte for output ;Check if 00-byte terminator
03166 03167	07F0 D5 07F1 7E 07F2 B7	DB\$MSG\$	Next: MOV ORA	- A, M A	x	;Check if OO-byte terminator
03166 03167 03168	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008	DB\$MSG\$	Next: MOV ORA JZ	- A, M A DB\$MSG\$	x	;Check if OO-byte terminator ;Exit
03166 03167 03168 03169	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23	DB\$MSG\$	Next: MOV ORA JZ INX	- A,M DB\$MSG\$ H	x	;Check if 00-byte terminator ;Exit ;Update message pointer
03166 03167 03168 03169 03170	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5	DB\$MSG\$	Next: MOV ORA JZ INX PUSH	- A,M DB\$MSG\$ H H	x	;Check if 00-byte terminator ;Exit ;Update message pointer ;Save updated pointer
03166 03167 03168 03169	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23	DB\$MSG\$	Next: MOV ORA JZ INX	- A,M DB\$MSG\$ H	x	;Check if 00-byte terminator ;Exit ;Update message pointer
03166 03167 03168 03169 03170 03171	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F8 4F	DB\$MSG\$	Next: MOV ORA JZ INX PUSH MOV	- A,M DB\$MSG\$ H H C,A		;Check if 00-byte terminator ;Exit ;Update message pointer ;Save updated pointer
03166 03167 03168 03169 03170 03171 03172	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F8 4F 07F9 CD0A05	DB\$MSG\$	Next: MOV ORA JZ INX PUSH MOV CALL	- A, M A DB\$MSG\$ H H C, A DB\$CONO		;Check if 00-byte terminator ;Exit ;Update message pointer ;Save updated pointer ;Ready for output
03166 03167 03168 03169 03170 03171 03172 03173	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F8 4F 07F9 CD0A05 07FC E1	DB\$MSG\$	Next: MOV ORA JZ INX PUSH MOV CALL POP	- A,M DB\$MSG\$ H H C,A DB\$CONO H	UT	Check if 00-byte terminator Fxit Update message pointer Save updated pointer Ready for output Recover message pointer
03166 03167 03168 03169 03170 03171 03172 03173 03174	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F8 4F 07F9 CD0A05	DB\$MSG\$	Next: MOV ORA JZ INX PUSH MOV CALL	- A, M A DB\$MSG\$ H H C, A DB\$CONO	UT	;Check if 00-byte terminator ;Exit ;Update message pointer ;Save updated pointer ;Ready for output
03166 03167 03168 03169 03170 03171 03172 03173	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F8 4F 07F9 CD0A05 07FC E1	DB\$MSG\$	Next: MOV ORA JZ INX PUSH MOV CALL POP	- A,M DB\$MSG\$ H H C,A DB\$CONO H	UT	Check if 00-byte terminator Fxit Update message pointer Save updated pointer Ready for output Recover message pointer
03166 03167 03168 03169 03170 03171 03172 03173 03174 03175	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F8 4F 07F9 CD0A05 07FC E1	;	Next: MOV ORA JZ INX PUSH MOV CALL POP JMP	- A,M DB\$MSG\$ H H C,A DB\$CONO H	UT	Check if 00-byte terminator Fxit Update message pointer Save updated pointer Ready for output Recover message pointer
03166 03167 03168 03169 03170 03171 03172 03173 03174 03175 03176	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F7 E5 07F9 CD0A05 07F9 CD0A05 07FD C3F107	DB\$MSG\$; DB\$MSG\$	Next: MOV ORA JZ INX PUSH MOV CALL POP JMP	A, M A DB\$MSG\$ H C, A DB\$CONO H DB\$MSG\$	UT	Check if 00-byte terminator FXit VDdate message pointer Save updated pointer Ready for output Recover message pointer Go back for next character
03166 03167 03168 03169 03170 03171 03172 03173 03174 03175 03176 03177	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F8 4F 07F9 CD0A05 07FC E1 07FD C3F107 0800 D1	;	Next: MOV ORA JZ INX PUSH MOV CALL POP JMP X: POP	A, M A DB\$MSG\$ H H C, A DB\$CONO H DB\$MSG\$ D	UT	Check if 00-byte terminator Fxit Update message pointer Save updated pointer Ready for output Recover message pointer
03166 03167 03168 03169 03170 03171 03172 03173 03174 03175 03176	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F7 E5 07F9 CD0A05 07F9 CD0A05 07FD C3F107	;	Next: MOV ORA JZ INX PUSH MOV CALL POP JMP	A, M A DB\$MSG\$ H C, A DB\$CONO H DB\$MSG\$	UT	Check if 00-byte terminator FXit VDdate message pointer Save updated pointer Ready for output Recover message pointer Go back for next character
03166 03167 03168 03169 03170 03171 03172 03173 03174 03175 03176 03177	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F8 4F 07F9 CID0A05 07FC E1 07FD C3F107 0800 D1 0801 C1	;	Next: MOV ORA JZ INX PUSH MOV CALL POP JMP X: POP POP	A, M A DB\$MSG\$ H C, A DB\$CONO H DB\$MSG\$ D B	UT	Check if 00-byte terminator FXit VDdate message pointer Save updated pointer Ready for output Recover message pointer Go back for next character
03166 03167 03168 03169 03170 03171 03172 03173 03174 03175 03176 03177	07F0 D5 07F1 7E 07F2 B7 07F3 CA0008 07F6 23 07F7 E5 07F8 4F 07F9 CD0A05 07FC E1 07FD C3F107 0800 D1	;	Next: MOV ORA JZ INX PUSH MOV CALL POP JMP X: POP	A, M A DB\$MSG\$ H H C, A DB\$CONO H DB\$MSG\$ D	UT	Check if 00-byte terminator FXit VDdate message pointer Save updated pointer Ready for output Recover message pointer Go back for next character

				
03180	0803 C9	RET		
03300	;#			
03301	7			
03302	;	Debug	input routine	
03303	,	Debdy	input reatine	
03304	;	This r	outine helps deb	ug code in which input instructions
03305				The opcode of the IN instruction
03306	;			value of OE7H (RST 4).
	•	must b	e replaced by a	Valde of CE/H (KS) 4/.
03307	;	T 1 1 1 1 1		ALC
03308	;			the port number contained in the byte
03309	;			converts it to hexadecimal, and
03310	;	displa	ys the message:	
03311	;			
03312	;		Input from por	
03313	;			
03314	7			aracters (in hex.) from the keyboard,
03315	;			ry in A, and then returns control
03316	;	to the	byte following	the port number
03317	;			
03318	;	*****		
03319	;			uses both DB\$CONOUT and BDOS calls
03320	;	*****	•	
03321	;			
03322	0804 496E707574DB			from Port 1
03323	0814 5858203A20DB	IN\$Port:	DB ^XX :	´,0
03324	;			
03325	;			
03326		\$Input:		
03327	081A 220904	SHLD	DB\$Save\$HL	;Save user's HL
03328	081D E1	POP	н	Recover address of port number
03329	081E 2B	DCX	н	Backup to point to RST
03330	081F 220F04	SHLD	DB\$Call\$Addres	s ;Save for later display
03331	0822 23	INX	н	;Restore to point to port number
03332				;Note: A need not be preserved
03333	0823 7E	MOV	A. M	;Get port number
03334	0824 23	INX	н	Update return address to bypass port number
03335	0824 23 0825 220D04	SHLD	DB\$Save\$RA	;Save return address
03336	0828 C5	PUSH	B	;Save remaining registers
03337	0829 D5	PUSH	ā	
03338	082A F5	PUSH		;Save port number for later
03339	002.1110			
03340				
03341	082B CDB108	CALL	DB&Flag\$Save\$	n ;Save current state of debug flag
03342	CC2B CDB100	UNCE	DDT: Tagtoarete	; and enable debug output
03343				; and enable debug od(pd(
03344	082E CDC107	CALL	DB&CRLE	;Display carriage return, line feed
03345	0831 CD7C07	CALL	A	LA; Display call address
03346	0834 F1	POP	PSW	Recover port number
03346	0835 211408	LXI	⊢sw H,DBIN\$Port	Recover port number
03348	0838 CDA307	CALL	DB\$CAH	Convert to hex.and store in message
03349	083B 210408	LXI	H, DBIN\$Message	;Output prompting message
03350	083E CDEE07	CALL	DB\$MSG	
03351	0841 0E02	MVI	C,2	;Get 2 digit hex, value
03352	0843 CDCF08	CALL	DB\$GHV	Returns value in HL
03353	0846 7D	MOV	A,L	;Get just single byte
03354	0047 000500	 .		
03355	0847 CDBF08	CALL	npat redakes to	e ;Restore debug output to previous state
03356	0044 54	000		Deserves used about
03357	084A D1	POP	D	Recover registers
03358	084B C1	POP	B	
03359	084C 2A0904	LHLD	DB\$Save\$HL	;Get previous HL
03360	084F E5	PUSH	H	;Put on top of stack
03361	0850 2A0D04		DB\$Save\$RA	;Get return address
03362	0853 E3	XTHL		;TOS = return address, HL = previous value
03363	0854 09	RET		
03500	;*			
03501	;			
03502	;	Debug	output routine	
03503	;			
03504	;			oug code in which output instructions
03505	;			The opcode of the OUT instruction
03506	;	must b	e replaced by a	value of OEFH (RST 5).
03507	;			
03508	;	This r	outine picks up	the port number contained in the byte
03509	;	follow	ing the RST 5, c	converts it to hexadecimal, and
03510	;		ys the message:	
03511	;			
L				

03512		;	Output to por	t XX : AA
03513		;		
03514				ents of the A register prior to the
03515			eing executed.	
03516 03517		; Control	is then retur	ned to the byte following the port number.
03518		, • *******		
03519			- This routin	e uses both DB\$CONOUT and BDOS calls
03520		; ******		
03521		;		
03522 03523	0855 4F75747075	; DBO#Massassa	DB 'Oute	white David (
03523	0864 5858203A20		DB 'XX:	put to Port 1
03525		DBO\$Value:	DB AA',	
03526		;		
03527		;		
03528		BB\$Output:	DD40	
03529 03530	086C 220904 086F E1	SHLD POP	DB\$Save\$HL H	;Save user's HL ;Recover address of port number
03531	0870 2B	DCX	Н	;Backup to point to RST
03532	0871 220F04	SHLD	DB\$Call\$Addre	ess ;Save for later display
03533	0874 23	INX	н	Restore to point at port number
03534	0875 324B04	STA	DB\$Save\$A	Preserve value to be output
03535	0878 7E	MOV	A, M	;Get port number
03536	0879 23 087A 220D04	INX SHLD	H DB\$Save\$RA	Update return address to bypass port number
03537 03538	087D C5	PUSH	DB#Save#KA B	;Save return address ;Save remaining registers
03539	087E D5	PUSH	D	youve remaining registers
03540	087F F5	PUSH	PSW	;Save port number for later
03541				
03542	0880 CDB108	CALL	DB\$Flag\$Save\$	On ;Save current state of debug flag
03543				; and enable debug output
03544 03545	0883 CDC107	CALL	DB\$CRLF	Display asyrisms ysturn line faad
03546	0886 CD7C07	CALL		;Display carriage return, line feed NLLA;Display call address
03547	0889 F1	POP	PSW	;Recover port number
03548	088A 216408	LXI	H,DBO\$Port	,
03549	088D CDA307	CALL	DB\$CAH	;Convert to hex.and store in message
03550				
03551 03552	0890 3A4B04 0893 216908	LDA LXI	DB\$Save\$A H,DBO\$Value	Comment water to be autout
03553	0896 CDA307	CALL	DB\$CAH	;Convert value to be output ;Convert to hex.and store in message
03554	00/0 02/00/	ONEL	2240111	yconvert to next and store in message
03555	0899 215508	LXI	H,DBO\$Message	;Output prompting message
03556	089C CDEE07	CALL	DB\$MSG	
03557				
03558	089F CDBF08	CALL	DB\$Flag\$Resto	ore ;Restore debug flag to previous state
03559 03560	08A2 D1	POP	n	;Recover registers
03561	08A3 C1	POP	B	skecover registers
03562	08A4 2A0904	LHLD	DB\$Save\$HL	;Get previous HL
03563	08A7 E5	PUSH	н	;Put on top of stack
03564	08A8 2A0D04	LHLD	DB\$Save\$RA	;Get return address
03565	08AB E3	XTHL		;TOS = return address, HL = previous value
03566 03567	08AC 3A4B04 08AF C9	LDA RET	DB\$Save\$A	Recover A (NOTE: FLAG NOT RESTORED)
03700		;#		
03701		;		
03702		; DB\$Flag	\$Save\$On	
03703				used for DB\$IN/OUT.
03704				state of the debug control flag,
03705			and then enab UT output alway	les it to make sure that
03708		; DD#1N/0	of output aiwa	iys goes out.
03708	08B0 00	, DB\$Flag\$Previou	s: DB	0 ;Previous flag value
03709		;		· · · · · · · · · · · · · · · · · · ·
03710		DB\$F1ag\$Save\$On		
03711	08B1 F5	PUSH	PSW	;Save caller's registers
03712	08B2 3A0604	LDA	DB\$Flag	;Get current value
03713	0885 328008 0888 3EFF	STA MVI	DB\$Flag\$Previo A.OFFH	ous ;Save it ;Set flag
03715	0888 320604	STA	DB\$Flag	A DEC I LEY
03716	08BD F1	POP	PSW	
03717	08BE C9	RET		
03800		;*		
03801		;		

03802		; DB\$Flag	*Restore	
03803		; This ro	utine is only us	ed for DB\$IN/OUT.
03804			ores the debug c	ontrol flag, DB\$Flag, to
03805		; its for	mer state.	
03806		; DB\$Flag\$Restore		
03808	08BF F5	PUSH	PSW	
03809	08C0 3AB008 08C3 320604	LDA	DB\$Flag\$Previou	
03810	08C3 320604	STA	DB\$Flag	;Set debug control flag
03811	08C6 F1	POP	PSW	
03812	08C7 C9	RET		
03813		;		
03900		, ;#		
03901		;		
03902		; Gethex	. value	
03903		,		
03904		; This su	broutine outputs	a prompting message, and then reads p get a hexadecimal value.
03905		; the key ; It is s	omenhat simplist	ic in that the first non-hex value
03907				he maximum number of digits to be
03908				as an input parameter. If more than the
03909		; maximum		ed, only the last four are significant.
03910		;		
03911 03912			WARN	**************************************
03912		; ; DB\$GHV		the BDOS to perform a read console
03914				ful if you use this routine from
03915		; within	an executing BIO	5.
03916		**********	****	******************
03917		;		
03918		; Entryp	arameters	
03919		;		erminated message to be output
03920		;		erminated message to be output exadecimal digits to be input
03922		;		Exadecimal digits to be impat
03923		;		
03924		DB\$GHV\$Buffer:		;Input buffer for console characters
03925		DB\$GHV\$Max\$Cour		
03926	08C8 00	DB	0	;Set to the maximum number of chars.
03927 03928		DB\$GHV\$Input\$Co	unt.	; to be input
03929	0809 00		0	;Set by the BDOS to the actual number
03930				; of chars, entered
03931		DB\$GHV\$Data\$Byt		
03932	08CA		5	Buffer space for the characters
03933		;		
03934 03935		; DB\$GHV:		
03935	08CF 79	DB#GHV: MOV	A,C	;Get maximum characters to be input
03937	08D0 FE05	CPI	5	;Check against maximum count
03938	08D2 DAD708	JC	DB\$GHV\$Count\$0k	;Carry set if $A < 5$
03939	08D5 3E04	MVI	A,4	;Force to only four characters
03940	A007 000000	DB\$GHV\$Count\$0k		A Cost in manimum actual in include biotecos
03941	08D7 32C808 08DA CDEE07	STA CALL	DB\$GHV\$Max\$Cour DB\$MSG	t ;Set up maximum count in input buffer ;Output prompting message
03942	08DD 11C808		D.DB\$GHV\$Buffer	
03944	OBEO OEOA	MVI	C, B\$READCONS	;Function code
03945	08E2 CD0500	CALL	BDOS	· ~
03946				
03947	08E5 0E02	MVI	C, B\$CONOUT	;Output a line feed
03948 03949	08E7 1E0A 08E9 CD0500	MVI CALL	E,OAH BDOS	
03949	JOE 7 CD0300	UMLL	2003	
03951	08EC 210000	LXI	н, о	;Initial value
03952	08EF 11CA08	LXI	D,DB\$GHV\$Data\$E	
03953	08F2 3AC908	LDA	DB\$GHV\$Input\$Co	
03954	08F5 4F	MOV	C,A	;Keep count in C
03955	08F6 0D	DB\$GHV\$Loop: DCR	с	;Downdate count
03956	08F7 F8	RM		;Return when all done (HL has value)
03957	08F8 1A	LDAX	D.	;Get next character from buffer
03959	08F9 13	INX	D	;Update buffer pointer
03960	08FA CD1B09	CALL	DB\$A\$To\$Upper	;Convert A to uppercase if need be
03961	08FD FE30	CPI	101	;Check if less than O
03962	08FF D8 0900 FE3A	RC CPI	<u> </u>	;Yes, terminate ;Check if > 9
03963	0900 FE3A 0902 DA1009	JC	DB\$GHV\$Hex\$Digi	
1				• · · · •

Figure 10-2. (Continued)

	03965	0905	FE41		CPI	'A'	;Check if < 'A'	
	03966	0907	D8		RC		;Yes, terminate	
	03967	0908	FE47		CPI	'F' + 1	;Check if > 'F'	
	03968	090A	DO		RNC		;Yes, terminate	
	03969	090B	D637		SUI	'A' - 10	Convert A through F to numeric	
	03970	090D	C31209		JMP	DB\$GHV\$Shift\$Le	ft\$4 :Combine with current result	
	03971			;			,	
	03972			DB\$GHV\$	Hex\$Diqi	t :		
	03973	0910	D630		SUI	101	:Convert to binary	
	03974			DB\$GHV\$	Shift\$Lei	ft\$4:	,,	
	03975	0912	29		DAD	н	:Shift HL left four bits	
	03976	0913	29		DAD	н		
	03977	0914	29		DAD	н		
1	03978	0915	29		DAD	н		
	03979	0916	85		ADD	L	Add binary value in LS 4 bits of A	
	03980	0917	6F		MOV	L,A	Put back into HL total	
	03981	0918	C3F608		JMP	DB\$GHV\$Loop	;Loop back for next character	
	04100			;#				
	04101			;				
	04102			;	A to upp	per		
	04103			;	Converte	s the contents o	of the A register to an uppercase	
	04104			;	letter :	if it is current	ly a lowercase letter	
	04105			.;				
	04106			;	Entry pa	arameters		
	04107			;				
	04108			;		A = character t	o be converted	
	04109			;				
	04110			;	Exit par	rameters		
	04111			;				
	04112			;		A = converted c	haracter	
	04113			;				
	04114			DB\$A\$To				
	04115	091B			CPI	'a'	;Compare to lower limit	
	04116	091D			RC		;No need to convert	
	04117	091E			CPI	′z′ + 1 .	;Compare to upper limit	
	04118	0920			RNC		;No need to convert	
	04119	0921			ANI	5FH	;Convert to uppercase	
	04120	0923	C9		RET			

Figure 10-2. Debug subroutines (continued)

```
B>ddt fig10-2.hex<cr>
DDT VERS 2.0
DT VERS 2.0
NEXT PC
0924 0000
=g100ccr2
0116 : Flags : Flags = C1Z0M1E1I0
0120 : A Register : A = AA
012F : B Register : B = BB
013E : C Register : C = CC
014D : D Register : C = CC
014D : D Register : L = D
015C : E Register : L = FF
015B : H Register : L = 11
0189 : Memory Dump #1 : Start, End Address : 0108, 0128
0108 : 05 3E AA 01 CC BB 11 EE : .>*. L;.n
0110 : DD 21 11 FF B7 37 CD 52 05 00 46 6C 61 67 73 00 : ]!.. 77MR ..F1 ags.
0120 : C1 52 05 02 41 20 52 65 67 : MR.. A Re g
01A0 : Memory Dump #2 : Start, End Address : 0100, 011F
0100 : 31 6B 03 CD EA 04 CD 15 05 3E AA 01 CC BB 11 EE : 1k.M j.M. .>*. L;.n
0110 : DD 21 11 FF B7 37 CD 52 05 00 46 6C 61 67 73 00 : ]!.. 77MR ..F1 ags.
01B7 : Memory Dump #3 : Start, End Address : 0101, 0100
** ERROR - Start Address > End **
01CE : Memory Dump #3 : Start, End Address : 0100, 0100
0100 : 31 : 1
```

Figure 10-3. Console output from debug testbed run

01E5 : BC Register : BC = BBCC
01F5 : DE Register : DE = DDEE
0205 : HL Register : HL = FF11
0215 : SP Register : SP = 0369
022E : Byte at (BC) : (BC) = BC
023F : Byte at (DE) : (DE) = DE
0250 : Byte at (HL) : (HL) = F1
026A : Word at (BC) : (BC+1),(BC) = $0B0C$
027B : Word at (DE) : (DE+1),(DE) = ODOE
028C : Word at (HL) : (HL+1),(HL) = 0F01
Debug output has been re-enabled.
This message should display 5 times
032B : Input from Port 11 : aa
032D : Output to Port 22 : AA

Figure 10-3. Console output from debug tested run (continued)

containing all of the symbols in your program, along with their respective addresses. Once the program has been loaded by SID, you can refer to the memory image of your program not by address, but by the actual symbol name from your source code. SID also supports the "pass count" concept when using breakpoints.

ZSID (Z80 Symbolic Debug)

This is the Z80 CPU's version of SID. The mini-assembler/disassembler uses Zilog instruction mnemonics rather than those used by Intel.

Bringing Up CP/M for the First Time

It is much harder to bring up CP/M on a new computer system than to debug an enhanced version on a system already running CP/M. You will often find yourself staring at a programmatic "brick wall" with no adequate debugging tools to assist you.

For example, you install the CP/M system on a diskette (using another CP/Mbased computer system), put the diskette into the new computer, and press the RESET button. The disk head loads on the disk, and then — nothing! You cannot use any programs such as DDT or SID because you do not yet have CP/M up and running on the new computer. Or can you?

The answer is, wherever possible, debug the code for the new machine on an existing CP/M system. You may have to "fake" some aspects of the new bootstrap or BIOS so that the act of testing it on the host machine does not interact with the CP/M already running on it.

This scheme permits you to be fairly sure of your program logic before loading the diskette into the new machine. It will help pin down problems caused by hardware problems on the new computer. The hardest situation of all is if you have only the new computer and the release diskettes from Digital Research. Your only option is to find a way of reading the CP/M image on the release diskette into memory, hand patch in new console and disk drivers (not a trivial task), write the patched image back onto a diskette, and resort to Orville Wright testing.

If you value your time, it is always more cost-effective to use another system with CP/M already installed. This is true even if the two systems do not have the same diskette format. You can still do the bootstrap and build the CP/M image on the host machine. Then download the image directly into the memory of the new machine and write it out to a diskette.

This *downloading* process does require, however, that the new computer have a read-only memory (ROM) monitor program. Depending on the capability of this ROM monitor program, you may have to hand patch into the new machine's memory a primitive "download" program that reads 8-bit characters from a serial port, stacking them up in memory and returning control to the monitor program when you press a keyboard character on the new machine's console. In fact, some ROM monitor programs have a downloading program built in.

Debugging the CP/M Bootstrap Loader

The CP/M bootstrap loader, as you may recall, is written on one of the outermost tracks on a diskette or hard disk. On a standard 8-inch single-sided, single-density diskette, CP/M's bootstrap loader is stored on the first sector of the first track. The loader is brought into memory by firmware that gets control of the CPU when you turn your machine on or press the RESET button.

The bootstrap has to be compact, as the diskette space on which it is stored is limited: no more than 128 bytes for standard 8-inch diskettes. This tends to rule out the use of the debug subroutines already described, so you have to fall back to more primitive techniques.

Testing the Bootstrap Under CP/M

A bootstrap is best developed on a CP/M-based system. The task is easiest of all if you already have CP/M running on your new machine and are simply preparing an enhanced version of the bootstrap loader. In this case, you can test most of the code as though it were a user program running in the transient program area (TPA).

Most bootstraps get loaded into memory at location 0000H, so at the front of the code to be debugged you must put a temporary origin line that reads

ORG 100H

If you omit this and ask DDT to load the HEX file output by the assembler, it will load at the true origin, 0000H, and wipe out the contents of the base page for the version of CP/M that you are running. This will cause a system crash; you will have to press the RESET button and reload CP/M. When this happens, DDT does not tell you directly that anything is amiss; it just displays a "?" after your request to load the HEX file. You will discover that the system has "gone away" only when you try to do something else.

You also will need to adjust the addresses into which the bootstrap tries to load the CP/M image. If you do not, you will overwrite the version of CP/M presently running.

With these adjustments made, you can load the bootstrap under DDT and watch it execute, confirming that it does load the correct image into the correct addresses for debugging and transfer control to the BIOS jump vector. When everything appears to be functioning correctly, use the IF instruction to disable the debug code, reassemble the bootstrap, and write it onto a diskette. Then put the diskette into drive A and press RESET.

Was the Bootstrap Loaded?

At this point you must establish whether the bootstrap is being loaded into memory when the machine is turned on or RESET is pressed. The best way of doing this, and one that you can leave in place permanently, is to output a sign-on message as soon as the loader gets control. This requires hardware set up to prepare the USART (Universal Synchronous/Asynchronous Receive/Transmit) chip to output data, although some manufacturers write this initialization code into the firmware that loads the bootstrap. A suitable sign-on message would be the following:

CP/M Bootstrap Loader : Vn 1.0 11/18/82

If you do not see this message, assume that control is *not* being transferred to the bootstrap loader. This will be useful in the future if someone should call you with a complaint that CP/M cannot be loaded. If this message does not appear, they probably do not have CP/M on the disk.

Did the Bootstrap Load CP/M?

This is a harder question to answer than whether the bootstrap itself has been loaded, especially if the bootstrap loader sign-on is displayed and then the system crashes. A sign-on message early in the BIOS cold boot processing can confirm the correct transfer of control into the BIOS.

If the problems with the bootstrap program are severe, you may have to adapt the memory-dump debugging subroutine, dumping the contents of memory to the console in order to see what information the bootstrap loader is placing in memory. Display 100H bytes starting from the front of the BIOS jump vector. This table has an immediately recognizable pattern of 0C3H values every three bytes. You should also check to see that the bootstrap is loading the correct number of sectors from the disk into memory. If it loads too few, CP/M may sign on only to crash a few moments later because it attempts either to execute code or access a constant at the end of the BIOS. If the bootstrap loads too many sectors from the disk, the excess may "wrap around" the top of memory and overwrite the bootstrap itself, down at location 0000H, before it has completed its task. In this case, you would see only the sign-on for the bootstrap, not for the BIOS.

Debugging the BIOS

Rather than try to debug the BIOS as a single piece of code, debug it as a series of separate functional modules.

Notwithstanding current "top-down" philosophies of dealing with overall structure first, it can be quicker to debug the low-level subroutines in a device driver first. This gives you a solid base on which to build.

The BIOS can be divided up into its constituent modules as follows:

- Character input Interrupt service Non-interrupt service
- Character output
- Interrupt routines Real time clock Watchdog timers
- Disk drivers High-level (deblocking) Low-level (physical I/O)

Plan to write a *testbed* program for each of these modules. This testbed code serves two purposes; first, it provides a means of transferring control into the module under test in a controlled way. Second, it includes the necessary modules or dummy modules to "fool" the module under test into responding as if it were running in a complete BIOS under CP/M.

Using the testbed, you can check every part of the module's logic except the part that may be time-critical. Problems caused by timing, such as interrupts disabled for too long or code that is too slow or too fast for a particular peripheral controller chip, tend to show up only when you are testing on the final hardware and when you are running your new BIOS under CP/M.

What You Should Test for in the BIOS

Describing fully how to debug each module in the BIOS ould fill several books. Remember that you are trying to establish the *absence* of errors using a technique that, by its very nature, tends to show only their *presence*.

There are two basic approaches to debugging. One is the plodding method, checking every aspect of the code to ensure that every feature really does work. The second is to try to do something useful with the code.

Plan to use both. Start with the plodding method, testing each feature under control of the testbed until you are sure that it is working *in vitro*. When all of the BIOS modules have been tested individually, build a CP/M system and try to do some useful work with it. Trying to use the system for actual work testing *in vitro* can be a good test.

Feature Checklist

Make a list of the specific features included in the various BIOS modules. Then devise specific test sequences that will show that each of the features is working correctly.

The same testbed code can often test all of the features of a driver module. If it cannot, create a new testbed for the more exotic features.

Keep the testbed routines. Experience shows that they are most often needed shortly after you have erased them. Even after you have tested the BIOS, the testbed routines will come in handy if you decide to enhance a particular driver later on. You can extract the driver code from the BIOS, glue it together with the testbed, and test the new feature code in isolation from the BIOS.

The following sections show example testbeds for the various drivers, along with example checklists. These checklists were used to test the example BIOS routines shown in earlier chapters.

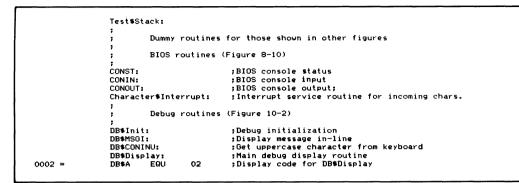
Character Drivers

Figure 10-4 shows the code for an example testbed routine for character I/O drivers in the BIOS. This code would be followed by the actual character I/O drivers, exactly as they would appear in the BIOS except that all IN and OUT instructions would be replaced with RST 4's and 5's respectively (see Figure 10-2) so that you could enter input values and inspect output values on the console.

This example contains the initialization code for the debug package shown in Figure 10-2 and the code setting up an RST 6 used to "fake" incoming character interrupts.

The main testbed loop consists of a faked incoming character interrupt followed by optional calls to CONIN or CONOUT, the return of control to DDT, or a loop back to fake another character interrupt. You can only return control to DDT if you used DDT to load the testbed and driver programs in the first place.

		;			/O drivers in the BIOS		
		;	The complete source file consists of three components:				
		;		1. The testbed			
		;		 The characte The debug pa 	r I/O drivers destined for the BIOS ckage shown in Figure 10-2.		
FFFF	=	; TRUE	EQU	OFFFFH			
0000		FALSE	EQU	NOT TRUE			
FFFF	=	DEBUG	EQU	TRUE	;For conditional assembly of RST ; instructions in place of IN and ; OUT instructions in the drivers		
0030	=	RST6	EQU	зон	;Use RST 6 for fake incoming character ; interrupt		
0100		START:	ORG	100H			
0100	31D101	0. HILL	LXI	SP, Test\$Stack	;Use a local stack		
	CDD101		CALL	DB\$Init	;Initialize the debug package		
0106	3EC3		MVI	A, JMP	;Set up RST 6 with JMP opcode		
0108	323000 21D101		STA	RST6			
010B	21D101 223100		LXI SHLD	H,Character\$Int RST6 + 1	errupt ;Set up RST 6 JMP address		
DICE	223100	;					
		;			character interrupt routine		
		;		re inal characie t buffer	rs can be captured and stored in		
		;					
	3EAA	Testbed	\$Loop: MVI	A, OAAH	;Set registers to known pattern		
	JEAA 01CCBB				;Set registers to known pattern		
				B, OBBCCH			
0110	11EEDD 2111FF		LXI LXI	D,ODDEEH H,OFF11H			
0110	E7		RST	6	;Fake interrupt for incoming character		
011D	CDD101	_		DB\$MSGI	;Display in-line message		
0120	0D0A456E74 4444542034	•	DB	'DDT : ',0	I to Input Char., O to Output, D to ente		
0152	4444542036	•	DB				
	CDD101		CALL	DB\$CONINU	;Get uppercase character		
015C	FE49		CPI	<1<	;CONIN?		
	CA7201		JZ	Go\$CONIN			
	FE44		CPI	1D1	;DDT?		
	CA6E01		JZ	Go\$DDT	0010170		
0166	FE4F CA9101		CPI JZ	101 Go\$CONOUT	; CONOUT?		
	C31101		JMP	Testbed\$Loop	;Loop back to interrupt again		
0100	031101	Go\$DDT:	ONE	lested*roob	FLOOP Back to Interrupt again		
016E	FF		RST	7	;Enter DDT (RST 7 set up by DDT)		
016F	C31101		JMP	Testbed\$Loop			
0172	CDD101	Go\$CONII	N: CALL	CONST	:Get console status		
	CA1101		JZ	Testbed\$Loop	No data waiting		
	CDD101		CALL	CONIN	;Get data from buffer		
017B	CDD101		CALL	DB\$Display	;Display character returned		
017E			DB	DB\$A	; in A register		
017F	434F4E494E	E	DB	'CONIN returned	´ ,0		
018E	C37201		JMP	Go\$CONIN	Repeat CONIN loop until no chars.		
		,			, waiting		
	000463	Go\$CONO					
	CDD101		CALL	CONST	;Get console status		
	CA1101		JZ	Testbed\$Loop	';No data waiting		
0197 019A	CDD101		CALL MOV	CONIN	Ready for output		
	CDD101		CALL	C,A CONOUT	;Ready for output ;Output to console		
	C39101		JMP	Go\$CONOUT	Repeat while there is still data		
		;					
	99999999999	>	DW		9H, 9999H, 9999H, 9999H, 9999H, 9999H		
01A1							
01A1 01B1	9999999999999		DW DW		9H, 9999H, 9999H, 9999H, 9999H, 9999H 9H, 9999H, 9999H, 9999H, 9999H, 9999H		





Executing an RST 7 without using DDT will cause a system crash, as DDT sets up the necessary JMP instruction at location 0038H in the base page.

The faked incoming character interrupt transfers control directly to the interrupt service routine in the BIOS (see the example in Figure 8-10, line 04902, label Character\$Interrupt). This reads the status ports of each of the character devices; you can enter the specific status byte values that you want. If you enter a value that indicates that a data character is "incoming," you will be prompted for the actual 8-bit data value to be "input." You can make the interrupt service routine appear to be inputting characters and stacking characters up in the input buffer. For debugging purposes, reduce the size of the input buffer to eight bytes. Making it larger means you will have to input more characters to test the buffer threshold logic. To check the interrupt service routine, you will pass through the main testbed loop doing nothing but faking incoming character interrupts and entering status and data values. The data characters will then be stacked up in the input buffer.

To check the correct functioning of the interrupt service routines, you can stay in control with DDT from the outset. Alternatively, you can just use DDT to load the testbed/driver HEX file, loop around inputting several characters, and then request that the testbed return control to DDT. Then you can use DDT to inspect the contents of the device table(s) and input buffers.

Another possibility is to create debugging routines that display the contents of the device table in a meaningful way, with each field captioned like this:

```
DEVICE TABLE O
     Status Port
                     81
                           Data Port
                                            80
                     01
                           Input Ready
                                            02
     Output Ready
     DTR high
                     40
     Reset Int. Prt D8
                          Reset Int. Val. 20
     2
     .
     Status Byte 1
          Output Suspended
          Output Xon Enabled
     :
```

```
:
Buffer Base OE8C
Put Offset 05 Get Offset 01
Char. Count 04 Control Count 00
Data Buffer
41 42 43 44 45 00 00 00
```

This display device table routine will require a fair amount of effort to code and debug—but it will pay dividends. You can obtain a complete "snapshot" of the device table without having to decode hexadecimal memory dumps and individual bits. Constant values in the device tables are also displayed, so that if a bug in your code corrupts the table, you will know about it immediately.

The next section shows examples of the specific tests you need to make, along with a description of the strategy you can use.

Interrupt Service Routine Checklist In a functioning BIOS, control is transferred to the interrupt service module whenever an incoming character causes an interrupt. In the example BIOS in Figure 8-10 (line 4900), the code scans each character device in turn to determine which one is causing the interrupt.

When you are debugging the interrupt service routines using the "fake" input/ output instructions, you will have to enter specific status byte values. Refer to the device table declarations in Figure 8-10, line 1500, to determine what values you must enter to make the service routine think that an incoming character is arriving or that data terminal ready (DTR) is high or low.

Start the debugging process using the first device table. Then repeat the tests on the other device tables.

The following is a checklist of features that should be checked in debugging the interrupt service routine:

Are all registers restored correctly on exit from the interrupt servicing?

Using DDT, start execution from the beginning of the testbed. Set a breakpoint (with the G100,nnnn command) to get control back immediately before the CALL Character\$Interrupt. Use the X command to display all of the registers, and then, by using the G,nnnn command, you set a breakpoint at the instruction that immediately follows the CALL Character\$Interrupt. The character drivers will prompt you for the status values. Enter 00 (which indicates that no character is incoming). Display the registers again — their values should be the same. Remember to check the value of the stack pointer and the amount of the stack area that has been used.

NOTE: Do not be too surprised if you lose control of the machine when you first try this test. You may have some fundamental logic errors initially. If the system crashes, reset it, reload CP/M, and then start the test again. This time, rather than setting the second breakpoint at the instruction following the CALL Character\$Interrupt, venture down into the Character\$Interrupt code and go through the code a few instructions

at a time, setting breakpoints before any instructions that could cause a transfer of control. Find out how far you are getting into the driver before it either jumps off into space or settles into a loop.

Does the service routine push a significant number of bytes onto the stack after an interrupt has occurred?

When you get control back after the CALL Character\$Interrupt, use the D (dump) command to dump the stack area's memory on the console. Check how far down the stack came by looking for the point where the constants that used to fill the stack area are overwritten by other data.

The example BIOS in Figure 8-10 saves only the contents of the HL register pair on the pre-interrupt stack. It then switches over to a private BIOS stack to save the contents of the rest of the registers and service the interrupt.

Are data characters added to the input buffer correctly?

"Input" a noncontrol character via the Character\$Interrupt routine. Then check the contents of the appropriate device table. The character count and the put offset should both be set to one. Then check the contents of the input buffer itself; does it contain the character that you "input?"

Are control characters added to the input buffer correctly?

"Input" a control character such as 01 H. Do not use ETX, ACK, XON, or XOFF (03H, 06H, 11H, and 13H, respectively); these may cause side effects if you have errors in the protocol handling logic. Check that the character is stored in the next byte of the input buffer and that the character and control counts are set to two and one, respectively. The put offset should also be set to two.

When the input buffer full threshold is reached, does the driver output the correct protocol character?

Set the first status byte in the first device table to enable input XON or RTS protocol, or both. Then go round the main testbed loop putting characters into the input buffer. Check the console display to see if the drivers output the correct values when the buffer is almost full (the default threshold is when five bytes remain). The driver should then drop the RTS line or output an XOFF character or both, according to the input protocol that you enabled.

When the input buffer is completely full, does the driver respond correctly?

This is an extension of the test above. Input one more character than can fit into the buffer. Check to see that the drivers do not stack the character into the input buffer and that a BELL character (07H) is output to the data port.

Are protocol characters XON/XOFF recognized and the necessary control flags set or reset?

Reload the testbed and drivers. Set the status byte to enable the output XON/XOFF protocol. Then use the Character\$Interrupt routine to input an XOFF character (13H). Check to see that the XOFF character has not been put into the input buffer. Instead, the status byte should be set to indicate that output has indeed been suspended.

Input an XON and check to see that the output suspended flag has been reset.

Does the driver detect and reset hardware errors correctly?

Proceed as though you were going to input a character into the input buffer, but instead enter a status byte value that indicates that a hardware error has occurred (enter the value given in the device table for DT\$Detect\$Error\$Value).

Check that the driver detects the error status and outputs the correct error-reset value to the appropriate control port.

Non-interrupt Service Routine Checklist In a "live" BIOS, non-interrupt service routines are accessed via the CONIN and CONST entry points in the BIOS jump vector. During debugging, the testbed can call the CONIN and CONST code directly.

Is input redirection functioning? Does control arrive in the driver with the correct device table selected?

This is best tested directly with DDT. Use the Gnnnn,bbbb command to transfer control into the CONIN code with a breakpoint at the RET instruction at the end of the Select\$Device\$Table routine (see Figure 8-10, line 04400). Check that the DE register pair is pointing at device table 0. If it is not, you will have to restart the test. Use the Tn command to make DDT trace through the Select\$Device\$Table subroutine to find the bug.

Are characters returned correctly from the buffer?

Use the testbed to "input" a character or two. Then use the testbed to make several entries into CONIN. Check the characters returned from the buffer.

Are the data character and control character counts correctly decremented?

After each character has been removed from the buffer by CONIN, use DDT to examine the device table and check that the data character and control character counts have been decremented correctly. Also check that the get pointer has moved up the input buffer.

When the buffer "almost empty" threshold is reached, does the driver emit the correct protocol character or manipulate the request to send (RTS) line correctly?

Use DDT to enable the input RTS or XON protocol or both. Then input characters into the input buffer until it reaches the buffer full threshold (the

default is when only five spare bytes remain in the buffer). Confirm that "buffer almost full" processing occurs. Then make repetitive calls to CONIN to flush data out of the buffer. Check that the "buffer emptying" processing occurs when the correct threshold is reached. For RTS protocol, the driver should output a raise RTS value to the specified RTS control port. For XON, the driver should output an XON character to the data port (after first having read the status port to ensure that the hardware can output the character).

Does the driver handle buffer "wraparound" correctly?

Input characters to the input buffer until it becomes completely full. Then make a single CONIN call to remove the first character from the buffer. Follow this by inputting one more character to the buffer. Check that the get pointer is set to one and the put pointer set to zero.

Next, make successive CONIN calls to empty the buffer. Then input one more character to the buffer. Check that this last character is put into the first byte of the input buffer.

Can the driver handle "forced input" correctly?

Using DDT, set the forced input pointer to point to a 00-byteterminated string; for example, use one of the function key decode default strings. (In Figure 8-10, the forced input pointer is initialized to point to a "startup string"—this is declared at the beginning of the configuration block at line 00400.)

Using DDT, call the CONST routine and check that it returns with A = 0FFH (indicating that there appears to be input data waiting).

Make successive calls to CONIN and confirm that the data bytes in the forced input string are returned. Check that the forcing of input ends when the 00H-byte is detected.

Does the console status routine operate correctly when it checks for data characters in the buffer, control characters in the buffer, and forced input?

Input a single noncontrol character, such as 41H, into the input buffer. Using DDT, check that the second status byte in the device table has the fake type-ahead flag set to zero. Call the CONST routine—it should return with A = 0FFH (meaning that there is data in the buffer). Then set the fake type-ahead bit in the second status byte and call CONST again. It should return with A = 00H (meaning that there is now "no data" in the buffer). Input a single control character into the buffer. Now CONST should return with A = 0FFH because there is a control character in the buffer.

Does the driver recognize escape sequences incoming from keyboard function keys?

This is a difficult feature to test when the real time clock routine is not running. The driver uses the watchdog timer to wait until all characters in the escape sequence have arrived. You will therefore have to modify the code in CONIN so that the watchdog timer appears to time out immediately, rather than waiting for the real time clock to tick. To make this change, refer to Figure 8-10, line 2200; this is the start of the CONIN routine. Look for the label CONIN\$Wait\$For\$Delay. A few instructions later there is a JNZ CONIN\$Wait\$For\$Delay. Using DDT, set all three bytes of this JNZ to 00H.

Then, using the testbed, input the complete escape sequence into the input buffer. For example, input hexadecimal values 1B, 4F, 51 (ESCAPE, O, P), which correspond to the characters emitted on a VT-100 terminal when FUNCTION KEY 1 (PF1) is pressed.

Next, use the testbed to make successive calls to CONIN. You should see the text associated with the function key (FUNCTION KEY 1, LINE FEED) being returned by CONIN.

Repeat this test using different function key sequences, including a sequence that does not correspond to any of the preset function keys. Check that the escape sequence itself is returned by CONIN without being changed into another string.

Can the driver differentiate between a function key and the same escape sequence generated by discrete key strokes?

This is almost the same test as above. Make the same patch to the CONIN code, only this time do not enter the complete escape sequence into the buffer. Enter only the hex characters 1B and 4F. Make sure that the CONIN routine does not substitute another string in place of this quasi-escape sequence.

This test only mimics the results of manually entering an escape sequence. You could not press the keys on a terminal fast enough to get all three characters into the input buffer within the time allowed by the watchdog timer.

Character Output Checklist Can the driver output a character?

The CONOUT option in the testbed calls CONIN first to get a character. To start with, you may want to use DDT to set the C register to some graphic ASCII character such as 41H (A), and transfer control into CONOUT directly. Check that CONOUT reads the USART's status, waits for the output ready value, and then outputs the data to the data port. Note that the testbed will output all characters waiting in the input buffer (or forced input) when you select its CONOUT option. This is a convenience for advanced testing of the drivers—for initial testing you may want to modify the testbed to make only one call to CONIN and CONOUT and then return to the top of the testbed loop. Does the driver suspend output when a protocol control flag indicates that output is to be suspended?

Using DDT, set the status byte in the device table to enable output XON/XOFF protocol. Then input an XOFF character and confirm that the output suspended bit in the status byte is set. Output a single character, and using DDT, confirm that the driver will remain in a status loop waiting for the output suspended bit to be cleared. Clear the bit using DDT and check that the character is output correctly.

When using ETX/ACK protocol, does the driver output an ETX after the specified number of characters have been output, then indicate that output is suspended?

For debugging purposes, alter the ETX message count value in the device table to three bytes. Then output three bytes of data via CONOUT. Check that the driver sends an ETX character (03H) after the three bytes have been output and that the output suspended flag in the status byte has been set.

Then input an ACK character (06H). Check that this character is not stored in the input buffer and that the output suspended flag is cleared.

Does the driver recognize and output escape sequences?

Input an ESCAPE, "t" (1BH, 74H) into the input buffer. Then output them via CONOUT. Using DDT, check that the CONOUT routine recognizes that an escape sequence is being output and selects the correct processing routine. In this case, the forced input pointer should be set to point at the ASCII time of day in the configuration block.

Does each of the escape sequence processors function correctly? Can the time and date be set to specified values using escape sequences?

Repeat the test above using all of the other escape sequences to make sure that they can be recognized and that they function correctly.

Real Time Clock Routines

A separate testbed program, shown in Figure 10-5, is used to check these routines. It calls the interrupt service routine directly to simulate a real time clock "tick," and then displays the time of day in ASCII on the console.

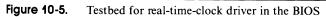
As you can see, the testbed makes a call into the debug package's initialization routine, DB\$Init, and then uses an RST 6 to generate fake clock "ticks."

There is a JMP instruction in the testbed that bypasses a call to Set\$Watchdog. Remove this JMP, either by editing it out or by using DDT to change it to NO OPERATIONS (NOP, 00H) when you are ready to test the watchdog routines.

Real Time Clock Test Checklist Is the clock running at all?

Using DDT, trace through the interrupt service routine logic. Check that the seconds are being updated.

		?	Testbed for real time clock driver in the BIOS.					
		; ;	The com	e consists of three components:				
		;	1. The testbed code shown here					
		;		 The real time clock driver destined for the BIOS. The debug package shown in Figure 10-2. 				
FFFF	=	; TRUE	EQU	OFFFFH				
0000		FALSE	EQU	NOT TRUE				
FFFF	=	DEBUG	EQU	TRUE	For conditional assembly of RST			
					<pre>; instructions in place of IN and ; OUT instructions in the drivers.</pre>			
0030	=	RST6	EQU	30H	;Use RST 6 for fake clock tick.			
0100			ORG	100H				
0100	318B01	START:	LXI	SP, Test\$Stack	;Use local stack			
	CD8B01		CALL	DB\$Init	;Initialize the debug package			
0106	3EC3		MVI	A, JMP	;Set up RST 6 with JMP opcode			
0108	323000 218B01		STA	RST6				
010B	218B01		LXI		;Set up RST 6 JMP address			
010E	223100		SHLD	RST6 + 1				
0111	C31D01		JMP	Testbed\$Loop	;<=== REMOVE THIS JMP WHEN READY TO ; TEST WATCHDOG ROUTINES			
0114	013200		LXI	B,50	;50 ticks before timeout			
	214201		LXI	H,WD\$Timeout	;Address to transfer to			
	CD8B01		CALL	Set\$Watchdog	;Set the watchdog timer			
		;		-				
		!	Make		TC intervent vouting			
		;			RTC interrupt routine correctly updated			
		;	to ensu	e that clock 15	consecty updated			
		, Testbed	Loop:					
011D			MVI	A, OAAH	;Set registers to known pattern			
	01CCBB		LXI	B, OBBCCH				
0122	11EEDD		LXI	D, ODDEEH				
0125	2111FF F7		LXI RST	H, OFF11H 6	:Fake interrupt clock			
				-				
	CD8B01 436C6F636I	3	CALL DB	DB\$MSGI 'Clock =',0	;Display in-line message			
0134	218B01		LXI	H,Time\$In\$ASCII	;Get address of clock in driver			
0137	CD8B01		CALL	DB\$MSG	;Display current clock value			
					; (Note: Time\$In\$ASCII already has			
			-		; a line feed character in it)			
013A 013D	CD8B01		DB	DB\$MSGI ODH, 0	;Display in-line message			
0130	0000		00	ODH, O	;Carriage return			
013F	C31D01	;	JMP	Testbed\$Loop				
		;		arrives here whe	n the watchdog timer times			
		;	out					
0142	CD8B01	WD\$Time@	CALL	DB\$MSGI				
	0D0A576174	L.	DB	ODH, OAH, 'Watchdo	na timed out1.0			
015A			RET	warry warehue	;Return to watchdog routine			
		;						
015B	99999999999	,	DW	9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H				
	99999999999999999999999999999999999999		DW	9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H				
01/8	7777777777	, Test\$Sta	DW ck:	7777n, 7777n, 9995	m, 7777m, 7777H, 7777H, 7777H, 7777H			
		;						
		;	Dummy routines for those shown in other figures BIOS routines (Figure 8-10)					
		;						
		;	BIOS LOUCTHES (LIGHTE O-IV)					
		, RTC\$Interrupt: ;Interrupt service routine for clock tick						
		Set\$Wate	hdog:	;Set wat	chdog timer			
		Time\$In4	ASCII:	;ASCII =	tring of HH:MM:SS, LF, O			
		; ;	Debug routines (Figure 10-2)					
		; DB\$Init:		Debug i	nitialization			
		DB\$MSGI			nitialization / message in-line			
		DB\$MSG:			message			



Are the hours, minutes, and seconds carrying over correctly?

Let the testbed code run at full speed. You should see the time being updated on the console display—although it will be updated much more rapidly than real time.

Use DDT to set the minutes to 58 and then let the clock run again. Does it correctly show the hour and reset the minutes to 00? Then set the hours to 11 and the minutes to 58 and let the clock run. Do minutes carry over into hours and are hours reset to 0?

Repeat these tests with the clock update constants set for 24-hour format.

Is the clock interrupt service routine restoring the registers correctly?

Using DDT, check that the registers are still set correctly on return from the clock interrupt service routine.

How much of a load on the pre-interrupt stack is the service routine imposing? Check the "low water mark" of the preset values remaining in the testbed stack area to see how much of a load the interrupt service routine is imposing on the stack.

Can the watchdog timer be set to a nonzero value? Can it be set back to zero? Using the second part of the testbed, call the Set\$Watchdog routine, and then monitor the testbed's execution as the watchdog timer times out. Check that the registers and stack pointer are set correctly when control is transferred to the timeout routine. Also check that control is returned properly from this routine, and thence from the interrupt service routine.

Disk Drivers

It is only feasible to check the low-level disk drivers in isolation from a real BIOS, as the BDOS interface to the deblocking code is very difficult to simulate. The testbed shown in Figure 10-6 serves only as a time-saver. It does not test the interface to the subroutines. Use DDT to set up the disk, track, and sector numbers, and then monitor the calls into SELDSK, SETTRK, SETSEC, SETDMA, and the read/write routines.

Unless you have the same disk controller on the host system as you do on the target machine, you will have to use the fake input/output system described earlier in this chapter, rather than attempt to read and write on real disks.

You can see that the testbed, after initializing the debugging package, makes calls to SELDSK, SETTRK, SETSEC, and SETDMA. It then calls a low-level read or write routine. The low-level routine called depends on which driver you wish to debug. For the standard floppy diskette driver shown in Figure 8-10, use Read\$No\$Deblock and Write\$No\$Deblock. For the 5 1/4-inch diskettes, use Read\$Physical and Write\$Physical. You will have to use DDT to set up some of the variables required by the low-level drivers that would normally be set up by the deblocking code.

Testbed for disk I/O drivers in the BIOS ; The complete source file consists of three components: : 1. The testbed code shown here . 2. The Disk I/O drivers destined for the BIOS 3. The debug package shown in Figure 10-2. . : FFFF = TRUE EQU OFFFFH 0000 = FALSE EQU NOT TRUE FFFF = DEBUG EQU TRUE ;For conditional assembly of RST ; instructions in place of IN and ; OUT instructions in the drivers. ORG 100H 0100 START: 0100 314704 LXI SP, Test\$Stack ;Use a local stack 0103 CD4704 DB\$Init CALL ;Initialize the debug package ; Make calls to SELDSK, SETTRK, SETSEC and SETDMA, ; then either a read or write routine. ; Testbed\$Loop: 0106 314704 LXI SP, Test\$Stack ;Use local stack 0109 3A1202 LDA Logical\$Disk ;Set up for SELDSK call 010C 4F 010D CD4704 MOV C, A SELDSK CALL 0110 CD4704 CALL DB\$Display ;Display return value in HL 0113 14 0114 53454C4453 DB DB\$HI 'SELDSK returned',0 DB 0124 223201 SHLD DPH\$Start ;Set up to display disk parameter header 0127 111000 LXI D,16 ;Compute end address 012A 19 DAD D DPH\$End 012B 223401 SHLD ;Store into debug call 012E CD4704 CALL DB\$Display ;Display DPH 0131 18 **DB** DR\$M ;Memory DPH\$Start: 0132 0000 DW 0 DPH\$End: 0134 0000 nω 0136 5365606563 'Selected DPH',0 DB LHLD Track 0143 2A1302 ;Call SETTRK 0146 E5 0147 C1 PUSH н POP в ;SETTRK needs track in BC 0148 CD4704 CALL SETTRK ;Call SETSEC 014B 3A1502 Sector 014E 4F 014F CD4704 MOV C,A SETSEC ;SETSEC need sector in C CALL 0152 011702 0155 CD4704 B,Test\$Buffer ;Set DMA address LXI SETDMA CALL 0158 3A1602 LDA Write\$Disk ;Check if reading or writing 015B B7 **ORA** 015C C2D101 JNZ Test\$Write 015E CD4704 CALL Read\$No\$Deblock ;*** or Read\$Physical depending on which ;*** drivers you are testing 0162 CD4704 CALL DB\$Display Display return code 0165 02 0166 5465737420 DB DB\$A 'Test Read returned',0 DB 0179 CD0102 017C CA0601 CALL Check\$Ripple ;Check if ripple pattern in buffer JZ Testbed\$Loop ;Yes, it is correct 017E CD4704 CALL DB\$MSGI ;Indicate problem DB\$HL ;Display HL (points to offending byte) 'Ripple pattern incorrect. HL -> failure.',0 0182 14 0183 526970706C **DB** DB\$HI DB 01AC CD4704 01AF CD1800 DB\$Display CALL :Display test buffer CALL DB\$M ; Memory 01B2 1702 DW Test\$Buffer

Figure 10-6. Testbed for disk I/O drivers in the BIOS

```
Test$Buffer$Size
01B4 0002
                           DW
01B6 436F6E7465
                           DB
                                      'Contents of Test$Buffer',0
01CE C30601
                           JMP
                                     Testbed$Loop
                  Test$Write:
01D1 CDF201
                                     Fill$Ripple ;Fill the test buffer with ripple pattern
Write$No$Deblock;*** or Write$Physical depending on which
;*** drivers you are testing
                           CALL
0104 004704
                           CALL
01D7 CD4704
                           CALL
                                     DB$Display
                                                        :Display return code
01DA 02
                           DB
                                     DB$A
01DB 5465737420
                                     'Test Write returned',0
                           DB
01EF C30601
                           JMP
                                     Testbed$Loop
                 Fill$Ripple:
                                                        ;Fills the Test$Buffer with a pattern
                                                        ; formed by putting into each byte, the
; least significant 8-bits of the byte's
                                                        :
                                                            address.
                                     B, Test$Buffer$Size
01F2 010002
                           IXI
01F5 211702
                           LXI
                                     H, Test$Buffer
                  FR$Loop:
                           MOV
01F8 75
                                                        ;Set pattern value into buffer
                                     M.L
01F9 23
                                                        ;Update buffer pointer
                           INX
                                     н
01FA OB
                           DCX
                                     в
                                                        ;Down date count
01FB 79
                           MOV
                                     A,C
                                                        ;Check if count zero
O1FC BO
                           ORA
                                     в
01FD C2F801
                           JNZ
                                     FR$Loop
                                                        ;Repeat until zero
0200 C9
                           RET
                  Check$Ripple:
                                                        :Check that the buffer is filled with the
                                                           correct ripple pattern
                                                           Returns with zero status if this is true,
nonzero status if the ripple is not
correct. HL point to the offending byte
                                                        :
                                                        :
                                                            (which should = L)
                                                        .
0201 010002
                                     B. Test$Buffer$Size
                           LXI
0204 211702
                           LXI
                                     H, Test$Buffer
                  CR$Loop:
                           MOV
0207 7D
                                     A.L
                                                        :Get correct value
0208 BE
                           CMP
                                     M
                                                        ;Compare to that in the buffer
0209 CO
                           RNZ
                                                         ;Mismatch, nonzero already indicated
020A 23
020B 0B
                            INX
                                     н
                                                         ;Update buffer pointer
                           DCX
                                     в
                                                        ;Downdate count
0200 79
                           MOV
                                     A,C
                                                        ;Check count zero
020D B0
                           ORA
                                     в
020E C20702
                                     CR$LOOP
                            JNZ
                                                        :Repeat until zero
0211 C9
                           RET
                                                        ;Zero flag will already be set
                  ;
                           Testbed variables
0212 00
                  Logical$Disk:
                                     DB
                                               n
                                                        ; A = 0, B = 1, \dots
0213 0000
                  Track:
                                     nω
                                               0
                                                        ;Disk track number
0215 00
                                                        ;Disk sector number
;NZ to write to disk
                  Sector:
                                     DB
                                               0
0216 00
                  Write$Disk:
                                     DB
                                               0
0200 =
                  .
Test$Buffer$Size
                                                        512
                                               EQU
                                                                 ;<=== Alter as required
                                     DS
                  Test$Buffer:
                                               Test$Buffer$Size
0217
0417 99999999999
0427 99999999999
                           DW
                                     9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H
                           DW
                                     9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H
0437 9999999999
                           DW
                                     9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H, 9999H
                  Test$Stack:
                           Dummy routines for those shown in other figures
                           BIOS routines (Figure 8-10)
                  SEL DSK:
                                               ;Select logical disk
                  SETTRK:
                                               ;Set track number
;Set sector number
;Set DMA address
                  SETSEC:
                  SETDMA:
                  Read$No$Deblock:
                                               ;Driver read routines
                  Read$Physical:
                  Write$No$Deblock:
                                               ;Driver write routines
                  Write$Physical:
```

Figure 10-6. (Continued)

```
;
                  ;
                           Debug routines (Figure 10-2)
                  DB$Init:
                                               ;Debug initialization
                  DB$MSGI:
                                              ;Display message in-line
;Main debug display routine
                  DB$Display:
0002 =
                  DBSA
                           FOU
                                     02
                                              ;Display codes for DB$Display
0014 =
                  DB$HL
                           EQU
                                     20
                  DB$M
                           EQU
                                     24
00'18 =
```

Figure 10-6. Testbed for disk I/O drivers in the BIOS (continued)

Before issuing the write call, the testbed fills the disk buffer with a known pattern. This pattern is checked on return from a read operation.

For both reading and writing, the testbed shows the contents of the A register. If you have added the enhanced disk error handling described in the previous chapter, the return value in A must *always* be zero.

Disk Driver Checklist Does SELDSK return the correct address and set up the required

system variables?

Check that the correct disk parameter header address is returned for legitimate logical disks. Check, too, that it returns an address of 0000H for illegal disks.

Check that any custom processing, such as setting the disk type and deblocking requirements from extra bytes on the disk parameter blocks, is performed correctly.

Does the SETTRK and SETSEC processing function correctly?

Using DDT, check that the correct variables are set to the specified values.

Does the driver read in the spare-sector directory correctly?

Set up to execute a physical read and, using DDT, trace the logic of the READ entry point. Check that the spare-sector directory would be loaded into the correct buffer. If you are using fake input/output, use DDT to patch in a typical spare-sector directory with two or three "spared-out" sectors.

Does the driver produce the correct spare sector in place of a bad one?

Continuing with the physical read operation, check that, for "good" track/sectors, the sector-sparing logic returns the original track and sector number, and for "bad" track/sectors, it substitutes the correct spare track and sector. If you are using sector skipping, check that the correct number of sectors is skipped.

Can a sector be read in from the disk?

Continuing further with the physical read, check that the correct sector is read from the specified disk and track. If you are using real I/O (as

opposed to faking it), the "ripple pattern" set by the testbed can be used, or you can fill the disk buffer area with some known pattern (using DDT's F command) so you can tell if any data gets read in.

Make sure you do not have any disks or diskettes in the computer system that are not write-protected—you may inadvertently write on a disk rather than read it during the early stages of testing.

Can a sector be written to the disk?

Using DDT, set up to write to a particular disk, track, and sector. Remove any write protection that you put on the target disk during earlier testing. You can either use the testbed's ripple pattern or fill the disk buffer area with a distinctive pattern. Write this data onto the disk, fill the buffer area with a *different* pattern, and read in the sector that you wrote. Check that the disk buffer gets changed back to the pattern written to the disk.

Does the driver display error messages correctly?

Rather than deliberately damaging a diskette to create errors, use DDT to temporarily sabotage the disk driver's logic. Make it return each of the possible error codes in turn, checking each time that the correct error message is displayed.

For each error condition in turn, check that the disk driver performs the correct recovery action, including interacting with the user and offering the choice of retrying, ignoring the error, or aborting the program.

Live Testing a New BIOS

Given that the drivers have passed all of the testing outlined above, you are ready to pull all of the BIOS pieces together and build a CP/M image.

For your initial testing, disable the real time clock, and use simple, polled I/O for the console driver if you can. It is important to get *something* up and running as soon as possible, and it is easier to do this without possible side effects from interrupts.

Prepare a complete listing of the BIOS and plan to spend at least an hour checking through it. Take a dry run through the console and disk driver — if there are any serious bugs left in these two drivers, CP/M may not start up. Remember that once the BIOS cold boot code has been executed and control is handed over to the CCP, the BDOS will be requested to log in the system disk, and this involves reading in the disk's directory.

Pay special attention to checking some of the major data structures. Make certain that everything is at a reasonable place in memory; for example, if the last address used by the BIOS is greater than 0FFFFH, you will need to move the entire CP/M image down in memory.

Then build a system disk, load it into the machine, and press the RESET button. You should see the bootstrap sign on, then the BIOS, and after a pause of about one second, the A> prompt (or 0A> if you have included the special feature that patches the CCP).

If you see both sign-on messages but do not get an A> prompt, a likely cause of the problem is in the disk drivers. Alternatively, the directory area on the disk may be full of random data rather than 0E5H's.

If you cannot see what is wrong with the system, you might try faking the disk drivers to return a 128-byte block of 0E5H's for each read operation. The CCP should then sign on.

Once you do have the A> prompt, you can proceed with the system checkout. Start by checking that the warm boot logic works. Type a CONTROL-C. There should be a slight pause, and the A> prompt should be output again.

Next, check that you can read the disk directory by using the DIR command. If you have an empty directory, you should get a NO FILE response. If you get strange characters instead, you either forgot to initialize the directory area or the disk parameter block is directing CP/M to the wrong part of the disk for the file directory. If the system crashes, there is a problem with the disk driver.

Check that you can write on the disk by entering the command SAVE 1 TEST. Then use the DIR command to confirm that file TEST shows up in the file directory. If it does, use the ERA command ERA TEST and do another DIR command to confirm that TEST has indeed been erased.

If TEST either does not show up on the disk or cannot be erased, then you have a problem with the disk driver WRITE routine.

Put a standard CP/M release diskette into drive B and use the DIR command to check that you can access the drive and display a disk directory. If you do, then load the DDT utility and exit from it by using a G0 (G, zero) command. This further tests if the disk drivers are functioning correctly.

To test the deblocking logic (if you are using disks that require deblocking), use the command:

PIP A:=B:*.*[V]

This copies all files from drive B to drive A using the verify option. It is a particularly good test of the system, and if you have any problems with the high-level disk drivers and deblocking code, you will get a Verify Error message from PIP. You can also get this message if you have hardware problems with the computer's memory, so run a memory test if you cannot find anything obviously wrong with the deblocking algorithm.

To completely test the deblocking code, you need to use PIP to copy a file of text larger than the amount of memory available. Thus, you may have to create a large text file using a text editor just to provide PIP with test data.

With the disk driver functioning correctly, rebuild the system with the real time clock enabled. Bring up the new system and check that the ASCII time of day is

being updated in the configuration block; use DDT to inspect this in memory. Set the clock to the current time, let it run for five minutes, and see if it is still accurate. You may have to adjust one of the initialization time constants for the device that is providing the periodic interrupts for the clock.

Rebuild the system yet again, this time with the real interrupt-driven console input and the real console output routines. Check that the system comes up properly and that the initial forced-input startup string appears on the console.

Check that when you type characters on the keyboard they are displayed as you type them. If not, there could be a problem with either the CONIN or CONOUT routines. Experimentally type in enough characters to fill the input buffer. If the terminal's bell starts to sound, the interrupt service routine is probably not the culprit. Check the CONOUT routine again.

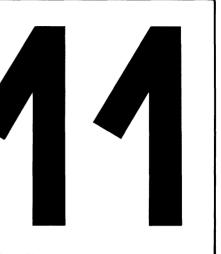
Check that the function key decode logic is working correctly. With the A> prompt displayed, press a function key. The CONIN driver should inject the correct function key string and it should appear on the terminal. For example, with the BIOS in Figure 8-10, pressing PF1 on the VT-100 terminal should produce this on the display:

```
A>Function Key1
Function?
A>
```

The CCP does not recognize "Function" as a legitimate command name, nor is there such a COM file—hence the question mark.

Using DDT, write a small program that outputs ESCAPE, "t" to the console, and check that the ASCII time of day string appears on the console. This checks that the escape sequence has been recognized.

Library Functions Reading or Writing Using the BIOS Accessing the File Directory Utility Programs Enhancing Standard CP/M Utility Programs for the Enhanced BIOS



Additional Utility Programs

This chapter contains the narrated source code for several useful utility programs. Two groups of such programs are included—those that supplement Digital Research's standard utility programs, and those that work in conjunction with features shown in the enhanced BIOS (Figure 8-10).

To avoid unnecessary detail, the programs shown in this chapter are all written in the C language. C is a good language to use for such purposes since it can show the overall logic of a program without the clutter of details common in assembly language.

In order to reuse as much source code as possible, this chapter includes a "library" of all the general-purpose C functions that can be called from within any of the utility programs. This file, called "LIBRARY.C", is shown in Figure 11-1. Once a utility program has been compiled, the necessary functions from the library can be linked with the utility's binary output to form the ".COM" file.

```
/* Library of commonly-used functions */
#include <LIBRARY.H>
                     /* Standard defines and structures */
       Configuration block access
                                     */
char
*get_cba(code)
                     /* Get configuration block address */
/*-----
/* This function makes a call to a "private" entry in the BIOS
jump vector to return the address of a specific data object in
the BIOS. The code indicates which object is required.
  Each program using this function could make a direct call to
  the BIOS using the biosh() function provided by BDS C. This
  function provides a common point to which debugging code can
  be added to display the addresses returned. */
/* Entry parameters */
                                                                          a
           /* Code that specifies the object
int code:
                   whose address is required */
/* Exit Parameters
  Address returned by the BIOS routine */
                     /* Value returned by the BIOS */
char *retval:
       retval = biosh(CBGADDR.code);
 /* printf("\nget_cba : code %d address %4x",code,retval); */
       return retval;
} /* End of get cba(code) */
/*
       Character manipulation functions
                                            */
/* String scan */
strscn(string.key)
/* This function scans a 00-terminated character string looking
  for a key string in it. If the key string is found within the
  string, the function returns a pointer to it. Otherwise it
   returns a value of zero. */
/* Entry parameters */
char *string;
char *key;
                     /* String to be searched */
                      /* Key string to be searched for */
/* Exit parameters
  Pointer to key string within searched string, or zero if key not found
¥/
                                                                          b
                      /* For all non-null chars. in string */
while (*string)
       if ((*string == *key) &&
                                     /* First char. matches */
           (sstromp(string,key) == 0) /* Perform substring
                                       compare on rest */
          Э
                                     /* Substring matches,
              return string:
                                        return pointer */
                                    /* Move to next char. in string */
       string++;
       3
return 0;
                                     /* Indicate no match found */
} /* End of strsen */
/* Uppercase string compare */
ustremp(string1,string2)
/* This function is similar to the normal strcmp function;
it differs only in that the characters are compared as if they
were all uppercase characters -- the strings are left
   unaltered. */
```

Figure 11-1. LIBRARY.C, commonly used functions, in C language

```
/* Entry Parameters */
char *string1;
char *string2;
                     /* Pointer to first string */
/* Pointer to second string */
/* Exit parameters
   0 - if string 1 = string 2
   -ve integer if string 1 > string 2
   +ve integer if string 1 < string 2
int count;
                        /* Used to access chars. in both strings */
                                                                               c
                        /* Start with the first character of both */
count = 0;
        /* While string 1 characters are non-null, and
           match their counterparts in string 2. */
while (string1[count] == string2[count])
        if (string1[++count] == (0^{\circ})^{\circ} /* Last char. in string 1 */
                                        /* Indicate equality */
                return 0;
return string2[count] - string1[count]; /* "Compare" chars. */
} /* End of sstremp */
sstrcmp(string,substring)
                                       /* Substring compare */
/* This function compares two strings. The first, string, need not
be 00-terminated. The second, substring, must be 00-terminated.
It is similar to the standard function stromp, except that the
   length of the substring controls how many characters are compared. */
/* Entry parameters */
char *string;
char *substring;
                        /* Pointer to main string */
                        /* Pointer to substring */
/* Exit parameters
   0 - substring matches corresponding characters in string -ve integer if char. in string is > char. in substring
   +ve integer if char. in string is < char. in substring
                                                                                d
int count;
                /* Used to access chars. in string and substring */
count = 0;
               /* Start with the first character of each */
        /* While substring characters are non-null, and
match their counterparts in string. */
while (string[count] == substring[count])
        if (substring[++count] == 1\01) /* Last char in substring */
                                        /* Indicate equality */
                return 0;
return substring[count] - string[count];
                                                /* "Compare" chars. */
} /* End of sstremp */
usstrcmp(string, substring) /* Uppercase substring compare */
/* This function compares two strings. The first, string, need not
   be 00-terminated. The second, substring, must be 00-terminated.
   It is similar to the substring compare above except all
   characters are made uppercase. */
                                                                                e
/* Entry parameters */
char *string;
                       /* Pointer to main string */
                        /* Pointer to substring */
char *substring:
/* Exit parameters
   0 -- substring matches corresponding characters in string
```

Figure 11-1. (Continued)

```
-ve integer if char. in string is > char. in substring
+ve integer if char. in string is < char. in substring
int count;
                /* Used to access chars in string and substring */
count = 0;
                /* Start with the first character of each */
                                                                                   e
        /* While substring characters are non-null, and
           match their counterparts in string. */
while (toupper(string[count]) == toupper(substring[count]))
        if (substring[++count] == (\0/) /* Last char. in substring */
                                         /* Indicate equality */
                return O:
        3
                                                 /* "Compare" chars. */
return substring[count] - string[count];
} /* End of usstrcmp */
comp fname(scb,name)
                                /* Compare file names */
      ______
/* This function compares a possibly ambiguous file name
   to the name in the specified character string. The number of
   bytes compared is determined by the number of characters in
   the mask.
   This function can be used to compare file names and types,
   or, by appending an extra byte to the mask, the file names,
   types, and extent numbers.
   For file directory entries, an extra byte can be prefixed to
   the mask and the function used to compare user number, file
   name, type, and extent.
Note that a "?" in the first character of the mask will NOT
   match with a value of 0xE5 (this value is used to indicate
   an inactive directory entry). */
/* Entry parameters */
                       /* Pointer to search control block */
/* Pointer to file name */
struct _seb *seb;
char *name:
/* Exit parameter
   NAME_EQ if the names match the mask
NAME_LT if the name is less than the mask
   NAME_GT if the name is greater than the mask
NAME_NE if the name is not equal to the mask (but the outcome
        is ambiguous because of the wildcards in the mask)
*/
                                                                                  f
int count;
                        /* Count of the number of chars. processed */
                        /* NZ when the mask is ambiguous */
/* Pointer to bytes at front of SCB */
short ambiguous;
char *mask;
/* Set pointer to characters at beginning of search control block */
mask = scb:
        /* Ambiguous match on user number, matches
only users 0 - 15, and not inactive entries */
if (mask[0] == '?')
        if (name[0] == 0xE5)
                return NAME NE; /* Indicate inequality */
e1se
        /* First char. of mask is not "?" */
        if (mask[0] != name[0]) /* User numbers do not match */
                return NAME_NE; /* Indicate inequality */
        7
/* No, check the name (and, if the length is such, the extent) */
     for (count = 1;
                                  /* Move to next character */
     count++)
        if (mask[count] == '?') /* Wildcard character in mask */
```

Figure 11-1. (Continued)

```
ambiguous = 1; /* Indicate ambiguous name in mask */
continue; /* Do not make any comparisons */
        if (mask[count] != (name[count] & Ox7F))
                { /* Mask char. not equal to FCB char. */
if (ambiguous) /* If previous wildcard, indicate NE */
                         return NAME_NE;
                else
                         /* Compare chars. to determine relationship */
                                                                                    f
                         return (mask[count] > name[count] ?
                                NAME_LT : NAME_GT);
                7
         /* If control reaches here, then all characters of the
        mask and name have been processed, and either there
        were wildcards in the mask, or they all matched. */
                         /* Indicate mask and name are "equal" */
return NAME EQ:
} /* End of comp_fname */
                 /* Convert file name for output */
conv_fname(fcb,fn)
      /*==
/* This function converts the contents of a file control
block into a printable string "D:FILENAME.TYP." */
/* Entry parameters */
struct _fcb *fcb;
char *fn;
                                 /* Pointer to file control block */
                                 /* Pointer to area to receive name */
        /* If the disk specification in the
g
*fn++ = ':':
                                         /* Insert disk id. delimiter */
movmem(&fcb -> fcb_fname,fn,8);
                                         /* Move file name
                                                             ¥۶
fn += 8;
*fn++ = '.';
                                         /* Update pointer */
/* Insert file name/type delimiter */
movmem(&fcb -> fcb_fname+8,fn,3);
                                         /* Move file type */
*fn++ &= 0x7F;
                                         /* Remove any attribute bits */
*fn++ &= 0x7F;
*fn++ &= 0x7F;
                                         /* Remove any attribute bits */
/* Remove any attribute bits */
*fn = '\0';
                                         /* Terminator */
} /* End of conv_fname */
conv_dfname(disk,dir,fn) /* Convert directory file name for output */
/* This function converts the contents of a file directory entry
block into a printable string "D:FILENAME.TYP," */
/* Entry parameters */
short disk;
                                /* Disk id. (A = 0, B = 1) */
                                 /* Pointer to file control block */
/* Pointer to area to receive name */
struct _dir *dir;
char *fn;
                                                                                    h
         /* Convert user number and disk id. */
sprintf(fn,"%2d/%c:",dir -> de_userno,disk + 'A');
fn += 5;
                                 /* Update pointer to file name */
movmem(&dir -> de_fname,fn,8); /* Move file name */
fn += 8;
*fn++ = '.';
                                 /* Update pointer */
/* Insert file name/type delimiter */
movmem(&dir -> de_fname+8,fn,3); /* Move file type */
*fn++ &= 0x7F;
                                 /* Remove any attribute bits */
*fn++ &= 0x7F;
                                 /* Remove any attribute bits */
*fn++ &= 0x7F;
                                 /* Remove any attribute bits */
/* Terminator */
*fn = '\0';
```

```
Ϋ́h
} /* End of conv dfname */
get_nfn(amb_fname,next_fname) /* Get next file name */
/* This function sets the FCB at "next_fname" to contain the
   directory entry found that matches the ambiguous file name
       "amb fname.
   in
   On the first entry for a given file name, the most significant
bit in the FCB's disk field must be set to one (this causes a
search first BDOS call to be made). */
/* Entry parameters */
/* Exit parameters
   0 = No further name found
   1 = Further name found (and set up in next fname)
×/
                         /* Set to either search first or next */
/* Pointer to file name in directory entry */
char bdos_func;
char *pfname;
/* Initialize tail-end of next file FCB to zero */
setmem(&next_fname -> fcb_extent,FCBSIZE-12,0);
                                                                                      i
bdos func = SEARCHF;
                         /* Assume a search first must be given */
if (!(next_fname -> fcb_disk & 0x80)) /* If not first time */
        /* search first on previous name */
srch_file(next_fname,SEARCHF);
        bdos_func = SEARCHN;
                                           /* Then do a search next */
         -
/* First time */
else
         next_fname -> fcb_disk &= 0x7F; /* Reset first-time flag */
         /* Refresh next_fname from ambiguous file name
            (move disk, name, type) */
movmem(amb_fname,next_fname,12);
/* If first time, issue search first, otherwise
issue a search next call. "srch_file" returns
a pointer to the directory entry that matches
the ambiguous file name, or 0 if no match */
if (!(pfname = srch_file(next_fname,bdos_func)) )
         ŧ
         return 0;
                         /* Indicate no match */
         3
/* Move file name and type */
movmem(pfname,&next_fname -> fcb_fname,11);
return 1;
                         /* Indicate match found */
} /* End of get_nfn */
char *srch_file(fcb,bdos_code) /* Search for file */
/* This function issues either a search first or search next
   BDOS call. */
/* Entry Parameters */
struct _fcb *fcb;
                                                                                      j
                        /* pointer to file control block */
/* either SEARCHF or SEARCHN */
short bdos_code;
/* Exit parameters
   0 = no match found
   NZ = pointer to entry matched (currently in buffer)
```

```
unsigned r_code;
                      /* Return code from search function
This is either 255 for no match, or 0, 1, 2, or 3
                          being the ordinal of the 32-byte entry in the
                         buffer that matched the name */
                       /* Pointer to directory entry */
char *dir entry;
       /* The BDS C compiler always sets the BDOS DMA to location 0x80 */
r_code = bdos(bdos_code,fcb);  /* Issue the BDOS call */
                                                                              j
if (r_code == 255)
                              /* No match found */
       return 0;
       /* Set a pointer to the matching
          entry by multiplying return code by 128
          and adding onto the buffer address (0x80), also add 1 to point to first character of name */
return (r code << 5) + 0x81;
}/* End of srch file */
rd disk(drb)
                     /* Read disk (via BIOS) */
/* This function uses the parameters previously set up in the
   incoming request block, and, using the BIOS directly,
executes the disk read. */
/* Entry parameters */
                      /* Disk request block (disk, track, sector, buffer) */
struct _drb *drb;
/* Exit parameters
  0 = No data available
  1 = Data available
                                                                              k
ŧ
if (!set_disk(drb))
                      /* Call SELDSK, SETTRK, SETSEC */
                    /* If SELDSK fails, indicate
       return 0;
                         no data available */
if (bios(DREAD))
                      /* Execute BIOS read */
                      /* Indicate no data available if error returned */
       return 0:
return 1;
                      /* Indicate data available */
} /* End of rd disk */
/* This function uses the parameters previously set up in the
  incoming request block, and, using the BIOS directly,
   executes the disk write. */
/* Entry parameters */
struct _drb *drb;
                     /* Disk request block (disk, track, sector, buffer) */
/* Exit parameters
  0 = Error during write
                                                                              L
  1 = Data written OK
                      /* Call SELDSK, SETTRK, SETSEC, SETDMA */
/* If SELDSK fails, indicate no data written */
if (!set_disk(drb))
       return O;
if (bios(DWRITE))
                      /* Execute BIOS write */
       return O;
                      /* Indicate error returned */
                      /* Indicate data written */
return 1;
} /* End of wrt_disk */
```

Figure 11-1. (Continued)

```
short set disk(drb)
                     /* Set disk parameters */
         ______________________
                                            ------
/* This function sets up the BIOS variables in anticipation of
   a subsequent disk read or write. */
/* Entry parameters */
                     /* Disk request block (disk, track, sector, buffer) */
struct _drb *drb;
/* Exit parameters
   0 = Invalid disk (do not perform read/write)
   1 = BIOS now set up for read/write
ж.
ŧ
       /* The sector in the disk request block contains a
          LOGICAL sector. If necessary (as determined by the
          value in the disk parameter header), this must be
          converted into the PHYSICAL sector.
          NOTE: skewtab is declared as a pointer to a pointer to
          a short integer (single byte). */
ewtab; /* Skewtab -> disk parameter header -> skew table */
short **skewtab:
                      /* Physical sector */
short phy_sec;
                                                                             m
       /* Call the SELDSK BIOS entry point. If this returns
          a O, then the disk is invalid. Otherwise, it returns
          a pointer to the pointer to the skew table */
if (!(skewtab = biosh(SELDSK,drb -> dr_disk)).)
return 0; /* Invalid disk */
bios(SETTRK.drb -> dr track); /* Set track */
       /* Note that the biosh function puts the sector into
          registers BC, and a pointer to the skew table in
          registers HL. It returns the value in HL on exit
          from the BIOS */
phy_sec = biosh(SECTRN,drb -> dr_sector,*skewtab); /* Get physical sector */
/* Indicate no problems */
return 1:
} /* End of setp disk */
/*
       Directory Management Functions
                                             */
get_nde(dir_pb)
                     /* Get next directory entry */
/* This function returns a pointer to the next directory entry.
   If the directory has not been opened, it opens it.
   When necessary, the next directory sector is read in.
  If the current sector has been modified and needs to be written back
onto the disk, this will be done before reading in the next sector. */
/* Entry parameters */
                            /* Pointer to the disk parameter block */
struct _dirpb *dir_pb;
/* Exit Parameters
   Returns a pointer to the next directory entry in the buffer.
                                                                             n
   The directory open and write sector flags in the parameter
   block are reset as necessary.
*/
if(!dir_pb -> dp_open)
                              /* Directory not yet opened */
       if (!open_dir(dir_pb)) /* Initialize and open directory */
               err dir(O DIR, dir pb);
                                            /* Report error on open */
               exit():
               /* Deliberately set the directory entry pointer to the end
                 of the buffer to force a read of a directory sector */
```

Figure 11-1. (Continued)

```
/* Reset write-sector flag */
/* Update the directory entry pointer to the next entry in
the buffer. Check if the pointer is now "off the end"
of the buffer and another sector needs to be read. */
if (++dir_pb -> dp_entry < dir_pb -> dp_buffer + DIR_BSZ)
        return dir_pb -> dp_entry;
                                            /* Return pointer to next entry */
        3
        /* Need to move to next sector and read it in */
         /* Do not check if at end of directory or move to
            the next sector if the directory has just been
            opened (but the opened flag has not yet been set) */
if (!dir_pb -> dp_open)
        dir_pb -> dp_open = 1; /* Indicate that the directory is now open */
else
        /* Check if the sector currently in the buffer needs to be
written back out to the disk (having been changed) */
        if (dir_pb -> dp_write)
                 Ŧ
                                               /* Reset the flag */
/* Write the directory sector */
                 dir_pb -> dp_write = 0;
if(!rw_dir(W_DIR,dir_pb))
                          err_dir(W_DIR,dir_pb); /* Report error on writing */
                          exit();
                          ł
                 3
                                                                                                 n
                 /* Count down on number of directory entries left to process,
        always four 32-byte entries per 128-byte sector */
dir_pb -> dp_entrem -= 4;
                 /* Set directory-end flag true if number of entries now < 0 */
        if (dir_pb -> dp_entrem == 0)
                                                    /* now at end of directory */
                 £
                 dir_pb -> dp_end = 1;
dir_pb -> dp_open = 0;
                                                    /* Indicate end */
/* Indicate directory now closed */
                 return 0;
                                                    /* Indicate no more entries */
                 /* Update sector (and if need be track and sector) */
        if (++dir_pb -> dp_sector == dir_pb -> dp_sptrk)
                 ŧ
                 ++dir_pb -> dp_track; /* Update track */
dir_pb -> dp_sector = 0; /* Reset sector */
        3
if(!rw_dir(R_DIR,dir_pb))
                                 /* Read next directory sector */
        err_dir(R_DIR,dir_pb); /* Report error on reading */
        exit();
        /* Reset directory-entry pointer to first entry in buffer */
return dir_pb -> dp_entry = dir_pb -> dp_buffer;
} /* End of get_nde */
open_dir(dir_pb) /* Open directory */
/* This function "opens" up the file directory
   on a specified disk for subsequent processing
by rw_dir, next_dir functions. */
                                                                                                 0
/* Entry parameters */
struct _dirpb *dir_pb; /* Pointer to directory parameter block */
```

```
ı
/* Exit parameters
   0 = Error, directory not opened
    1 = Directory open for processing
× /
struct _dpb *dpb;
                                           /* CP/M disk parameter block */
           /* Get disk parameter block address for the disk specified in
              the directory parameter block */
if ((dpb = get_dpb(dir_pb -> dp_disk)) == 0)
           return 0;
                                /* Return indicating no DPB for this disk */
           /* Set the remaining fields in the parameter block */
dir_pb -> dp_sptrk = dpb -> dpb_sptrk; /* Sectors per track */
dir_pb -> dp_track = dpb -> dpb_trkoff; /* Track offset of the directory */
dir_pb -> dp_sector = 0; /* Beginning of directory */
air_pb -> dp_track = dpb -> dpb_trkott; /* irack ottset of the directory */
dir_pb -> dp_sector = 0; /* Beginning of directory */
dir_pb -> dp_nument = dpb -> dpb_maxden+1; /* No. of directory entries */
dir_pb -> dp_entrem = dir_pb -> dp_nument; /* Entries remaining to process */
dir_pb -> dp_end = 0; /* Indicate not at end */
/* Set number of allocation blocks per directory entry to
8 or 16 depending on the number of allocation blocks */
dir_pb -> dp_nabpde = (dpb -> dpb_maxabn > 255 ? 8 : 16);
           /* Set number of allocation blocks (one more than number of
              highest block) */
dir_pb -> dp_nab = dpb -> dpb_maxabn;
           /* Set the allocation block size based on the block shift.
The possible values are: 3 = 1k, 4 = 2K, 5 = 4K, 6 = 8K, 7 = 16K.
So a value of 16 is shifted right by (7 - bshift) bits. */
dir_pb -> dp_absize = 16 >> (7 - dpb -> dpb_bshift);
                                /* Indicate that directory now opened */
return 1:
} /* End of open_dir */
rw_dir(read_op,dir_pb) /* Read/write directory */
                                     ______
                                                                   -----
/* This function reads/writes the next 128-byte
    sector from/to the currently open directory. */
/* Entry parameters */
short read_op; /* True to read, false (0) to write */
struct _dirpb *dir_pb; /* Directory parameter block */
/* Exit parameters
0 = error -- operation not performed
   1 = Operation completed
struct _drb drb;
                                           /* Disk request (for BIOS read/write) */
drb.dr_disk = dir_pb -> dp_disk;
                                                     /* Set up disk request */
drb.dr_track = dir_pb / dr_track;
drb.dr_track = dir_pb -> dr_track;
drb.dr_sector = dir_pb -> dr_sector;
drb.dr_buffer = dir_pb -> dp_buffer;
if (read_op)
           if (!rd_disk(&drb))
                                           /* Issue read command */
                                         /* Indicate error -- no data available */
                      return Op
           ł
else
                                          /* Issue write command */
           if (!wrt_disk(&drb))
                      return 0:
                                           /* Indicate error -- no data written */
return 1;
                                           /* Indicate operation complete */
} /* End of rd_dir */
```

0

р

Figure 11-1. (Continued)

```
/* This function displays an error message to report an error
   detected in the directory management functions open_dir and rw_dir. */
/* Entry parameters */
short opcode;
                               /* Operation being attempted */
struct _dirpb *dir_pb; /* Pointer to directory parameter block */
printf("\n\007Error during ");
switch(opcode)
        case R DIR:
                                                                               q
               printf("Reading");
               break;
        case W DIR:
               printf("Writing");
               break;
        case O_DIR:
               printf("Opening");
               break;
        default:
               printf("Unknown Operation (%d) on",opcode);
        3
printf(" Directory on disk %c:. ",dir_pb -> dp_disk + 'A');
} /* End of err_dir */
setscb(scb,fname,user,extent,length)
                                     /* Set search control block */
           /*=========
                                                     _____/
/* This function sets up a search control block according
   to the file name specified. The file name can take the
   following forms:
       filename
       filename.typ
       d:filename.typ
       *:filename.typ (meaning "all disks")
       ABCD...NOP:filename.typ (meaning "just the specified disks")
  The function sets the bit map according to which disks should be
  searched. For each selected disk, it checks to see if an error is
generated when selecting the disk (i.e. if there are disk tables
in the BIOS for the disk). */
/* Entry parameters */
                      /* Pointer to search control block */
/* Pointer to the file name */
struct _scb *scb;
char *fname;
                       /* User number to search for */
short user;
                                                                               r
                      /* Extent number to search for */
/* Number of bytes to compare */
short extent;
int length;
/* Exit parameters
  None.
×/
int disk:
                      /* Disk number currently being checked */
unsigned adisks;
                      /* Bit map for active disks */
adisks = 0:
                       /* Assume no disks to search */
if (strscn(fname,":"))
                               /* Check if ";" in file name */
        if (#fname == '#')
                              /* Check if "all disks" */
               £
               adisks = 0xFFFF;
                                      /* Set all bits */
               3
       else
                               /* Set specific disks */
               while(#fname != ':')
                                      /* Until ":" reached */
```

Figure 11-1. (Continued)

```
/* Build the bit map by getting the next disk
                           id. (A - P), converting it to a number in
the range 0 - 15, shifting a 1-bit left
that many places, and OR-ing it into the
current active disks. */
adisks != 1 << (toupper(*fname) - 'A');</p>
                                             /* Move to next character */
                           ++fname:
                           3
                                            /* Bypass colon */
                  ++fname;
         /* Use only current default disk */
0150
                  /* Set just the bit corresponding to the current disk */
         adisks = 1 << bdos(GETDISK);</pre>
setfcb(scb,fname);
                          /* Set search control block as though it
                              were a file control block. */
/* Make calls to the BIOS SELDSK routine to make sure that
   all of the active disk drives have disk tables for them
    in the BIOS. If they don't, turn off the corresponding
   bits in the bit map. */
                          /* Start with disk A: */
for (disk = 0)
     disk < 16;
                          /* Until disk P: */
                          /* Use next disk */
     disk++)
         £
         if ( !((1 << disk) & adisks))
                                            /* Avoid selecting unspecified disks */
/* Make BIOS SELDSK call */
                  continue;
         if (biosh(SELDSK,disk) == 0)
                                             /* Returns O if invalid disk */
                  ÷
                  /* Turn OFF corresponding bit in mask
                  by AND-ing it with bit mask having
all the other bits set = 1 */
adisks &= ((1 << disk) ^ 0xFFFF);</pre>
                  3
         3
scb -> scb_adisks = adisks; /* Set bit map in SCB */
scb -> scb_userno = user; /* Set user number */
scb -> scb_extent = extent; /* Set extent number */
scb -> scb_length = length; /* Set number of bytes to compare */
} /* End setscb */
dm_clr(disk_map) /* Disk map clear (to zeros) */
/* This function clears all elements of the disk map to zero. */
/* Entry Parameters */
unsigned disk_map[16][18];
                                 /* Address of array of unsigned integers */
/* Exit parameters
   None.
×/
£
         /* WARNING -- The 576 in the setmem call below is based on
            the disk map array being [16][18] -- i.e. 288 unsigned
integers, hence 576 bytes. */
setmem(disk_map,576,^\0'); /* Fill array with zeros */
} /* End of dm_clr */
/*-------*/
dm_disp(disk_map,adisks)
                                           /* Disk map display */
   _____
/* This function displays the elements of the disk map, showing
   the count in each element. A zero value-element is shown as
   blanks. For example:
```

r

s

t

Figure 11-1. (Continued)

```
9 10 11 12 13 14 15 Used Free
     Δ
         1
             2
                3
                     4 5
                            6
                                7
                                    8
            20 98
                          202
                                  199 101 211
A: 123
                                                                   954 70
   Lines will only be printed for active disks (as indicated by
   the bit map). */
/* Entry parameters */
unsigned disk_map[16][18];
                               /* Pointer to disk map array */
unsigned adisks;
                               /* Bit map of active disks */
#define USED_COUNT 16
                               /* "User" number for used entities */
/* "User" number for free entities */
#define FREE_COUNT 17
                               /* Current disk number */
int disk:
int userno:
                               /* Current user number */
unsigned dsum:
                               /* Sum of entries for given disk */
                                          7 8 9 10 11 12 13 14 15 Used Free");
printf("\n
               0 1 2 3 4 5 6
                               /* Start with disk A: */
/* Until disk P: */
for (disk = 0;
     disk < 16:
     disk++)
                               /* Next disk */
        £
        if (!(adisks & (1 << disk)))
                                      /* Check if disk is active */
                              /* No -- so bypass this one */
               continue:
       printf("\n%c: ",disk + 'A');
                                      /* Display disk number */
                                                                                              t
        dsum = 0;
                               /* Reset sum for this disk */
        for (userno = 0;
                              /* Start with user 0 */
            userno < 16;
                               /* Until user 15 */
            userno++)
                               /* Next user number */
                ş
               dsum += disk_map[disk][userno]; /* Build sum */
        if (dsum)
                       /* Check if any output for this disk,
and if not, display d: None */
                /* Print either number or blanks */
                                   /* Start with user 0 */
                for (userno = 0;
                    userno < 16;
                                       /* Until user 15 */
                    userno++)
                                       /* Next user number */
                       if (disk_map[disk][userno])
                               printf("%4d",disk_map[disk][userno]);
                       else
                               printf("
                                          ").
                       ł
               3
       else
                       /* No output for this disk */
               £
               printf( " -- None --
       printf("
                %4d %4d",disk_map[disk][USED_COUNT],disk_map[disk][FREE COUNT]);
} /* End dm_disp */
get dpb(disk)
                      /* Get disk parameter block address */
/* This function returns the address of the disk parameter
  block (located in the BIOS). */
/* Entry parameters */
char disk;
                      /* Logical disk for which DPB address is needed */
                                                                                              u
/* Exit parameters
       0 = Invalid logical disk
       NZ = Pointer to disk parameter block
*/
if (biosh(SELDSK,disk) == 0)
                                       /* Make BIOS SELDSK call */
       return 0;
                                       /* Invalid disk */
```

```
/* Use BDOS SETDISK function */
/* Get the disk parameter block */
    hdos(SFTDISK, disk);
    return bdos(GETDPARM):
                                                                                                 11
    } /* End of get_dpb */
             Code table functions
    /*
                                        ¥ /
    /* Most programs that interact with a user must
        accept parameters from the user by name and translate
        the name into some internal code value.
        They also must be able to work in reverse, examining
        the setting of a variable, and determing what (ASCII
        name) it has been set to.
        An example is setting baud rates. The user may want to enter "19200," and have this translated into a number
        to be output to a chip. Alternatively, a previously
        set baud rate variable may have to be examined and the string "19200" generated to display its current
        setting to the user.
        A code table is used to make this task easier.
        Each element in the table logically consists of:
             A code value (unsigned integer)
             An ASCII character string (actually a pointer to it) */
                                                                                                 v
    ct_init(entry,code,string)
                                       /* Initialize code table */
        /* This function initializes a specific entry in a code table
       with a code value and string pointer.
");
       NOTE: By convention, the last entry in a given code table will have a code value of CT_SNF (string not found). */
    /* Entry parameters */
    struct _ct *entry;
int code;
                                        /* Pointer to code table entry */
                                       /* Code value to store in entry */
                                        /* Pointer to string for entry */
    char *string:
    ⊷×it
None.
*/
    /* Exit parameters
    entry -> _ct_code = code;
entry -> _ct_sp = string;
                                                /* Set _ct_code */
/* Set string pointer */
    } /* end of ct_inti */
    unsigned
                                        /* Parameter - return code */
    ct_parc(table,string)
    /* This function searches the specified table for a
       matching string, and returns the code value that corresponds to it.
        If only one match is found in the table, then this function returns
        that code value. If no match or more than one match is found,
        it returns the error value, CT_SNF (string not found).
        This function is specifically designed for processing
        parameters on a command tail.
       Note that the comparison is done after conversion to uppercase
(i.e. "STRING" matches "string"). A substring compare is used so
that only the minimum number of characters for an unambiguous
response need be entered. For example, if the table contained:
                                                                                                 w
                      Code
                               Value
                               "APPLES"
                      1
                               "ORANGES"
                      2
                      2
                               "APRICOTS"
        A response of "O" would return code = 2, but "A" or "AP" would be ambiguous. "APR" or "APP" would be required. */
    struct _ct *table;
char *string;
                                        /* Pointer to table */
/* Pointer to key string */
```

Figure 11-1. (Continued)

```
int mcode;
                               /* Matched code to return */
int mcount;
                               /* Count of number of matches found */
mcode = CT SNF;
                               /* Assume error */
mcount = 0:
                               /* Reset match count */
while(table -> _ct_code != CT_SNF) /* Not at end of table */
        /* Compare keyboard response to table entry using
           uppercase substring compares */
        if (usstrcmp(table -> _ct_sp,string) == 0)
                                                                            w
                mcount++;
                              /* Update match count */
                mcode = table -> _ct_code;
                                            /* Save code */
                3
        table++;
                              /* Move to next entry */
        2
                              /* Only one match found */
if (mcount == 1)
       return mcode;
                              /* Return matched code */
/* Illegal or ambiguous */
else
        return CT_SNF;
} /* End ct_pare */
unsigned
ct_code(table,string) /* Return code for string */
_____*/
/* This function searches the specified table for the
   specified string. If a match occurs, it returns the
   corresponding code value. Otherwise it returns CT_SNF
   (string not found).
   Unlike ct_parc, this function compares every character in the
   key string, and will return the code on the first match found. */
/* Entry parameters */
/* Entry Perameter -
struct _ct *table; /* Pointer to ter-
/* Pointer to string */
                                                                            х
/* Exit parameters
   Code value -- if string found
CT_SNF -- if string not found
while(table -> _ct_code != CT_SNF)
                                     /* For all entries in table */
        if (ustrcmp(table -> _ct_sp,string) == 0) /* Compare strings */
    return table -> _ct_code; /* Return code */
table++; /* Move to next entry */
return CT_SNF;
                                      /* String not found */
} /* End ct code */
ct_disps(table) /* Displays all strings in specified table */
/* This function displays all of the strings in a given table.
   It is used to indicate valid responses for operator input. */
/* Entry parameters */
                             /* Pointer to table */
struct _ct *table;
/* Exit Parameters
                                                                           у
       None.
*/
while(table -> _ct_code != CT_SNF)
                                     /* Not end of table */
       printf("\n\t\t%s",table -> _ct_sp); /* Print string */
        table++;
                                      /* Move to next entry */
        3
```

386 The CP/M Programmer's Handbook

```
putchar((\n/):
                                         /* Add final return */
                                                                                    v
} /* End of ct disps */
ct_index(table,string) /* Returns index for a given string */
/* This function searches the specified table, and returns
   the INDEX of the entry containing a matching string.
All characters of the string are used for the comparison,
   after they have been made uppercase. */
/* Entry parameters */
struct _ct *table;
char *string;
                               /* Pointer to table */
/* Pointer to string */
/* Exit parameters
  Index of entry matching string, or
CT_SNF if string not found.
¥7
                                                                                    7
int index;
                                /* Current value of index */
                                /* Initialize index */
index = 0:
while(table -> _ct_code != CT_SNF)
                                       /* Not at end of table */
        if (ustrcmp(table -> _ct_sp,string) == 0)
    return index; /* Return index */
table++; /* Move to next table entry */
index++; /* Update index */
return CT_SNF;
                  /* String not found */
¥
/*======
/* This function returns a pointer to the string in the
   table entry specified by the index. */
/* Entry parameters */
struct_ct *table; /* Pointer to table */
int index; /* Index into table */
                                                                                    a a
       _ct *entry; /* Entry pointer */
entry = table[index]; /* Point to entry */
return entry -> _ct_sp; /* Return pointer to string */
struct _ct *entry;
} /* End of ct_stri */
char *ct_strc(table,code) /* Get string according to code value */
        /*======
/* This function searches the specified table and returns a
  pointer to the character string in the entry with the matching code value or a pointer to a string of "unknown" if the code value is not found. */
                                                                                    bb
/* Entry parameters */
                               /* Pointer to table */
struct _ct *table;
unsigned code;
                                /* Code value */
while(table -> _ct_code != CT_SNF)
                                        /* Until end of table */
        t
t
if (table -> _ct_code == code) /* Check code matches */
return table -> _ct_sp; /* Yes, return ptr. to str. */
table++;
/* No. move to next entry */
                                         /* No, move to next entry */
```

Figure 11-1. (Continued)

```
return "Unknown";
       Bit vector functions
/*
                             ×/
                                                                      bb
/# These functions manipulate bit vectors. A bit vector is a group
   of adjacent bits, packed eight per byte. Each bit vector has the structure defined in the LIBRARY.H file.
   Bit vectors are used primarily to manipulate the operating
   system's allocation vectors and other values that can best
be represented as a series of bits. */
bv_make(bv,bytes)
                    /* Make a bit vector and clear to zeros */
    ============×/
/* This function uses C's built-in memory allocation, alloc,
   to allocate the necessary amount of memory, and then
   sets the vector to zero-bits. */
/* Entry parameters */
struct _bv *bv;
unsigned bytes;
                     /* Pointer to a bit vector */
                     /* Number of bytes in bit vector */
/* Exit parameter
   NZ = vector created
   0 = insufficient memory to create vector
                                                                      сс
×.
if(!(bv -> bv_bits = alloc(bytes)))
                                    /* Request memory */
       return 0;
                                    /* Request failed */
                                    /* Set length */
bv -> bv_bytes = bytes;
bv -> bv_end = bv -> bv_bits + bytes; /* Set pointer to end */
bv fill(bv,0);
                                    /* Fill with O's */
return 1;
} /* End by_make */
bv_fill(bv,value) /* Fill bit vector with value */
/* This function fills the specified bit vector with the
   specified value.
   This function exist only for consistency's sake and to isolate the main body of code from standard functions like setmem. */
/* Entry parameters */
struct _bv *bv;
char value;
                     /∗ Pointer to bit vector ∗/
                                                                      d d
                     /* Value to fill vector with */
/* Exit parameters
  None.
¥/
/×
      address
                  length
                                value */
setmem(bv -> bv_bits,bv -> bv_bytes,value);
bv_set(bv,bitnum) /* Set the specified bit number */
/*_____
/* This function sets the specified bit number in the bit vector
to one-bit. */
                                                                      e e
/* Entry parameters */
                             /* Pointer to bit vector */
struct _bv *bv;
unsigned bitnum;
                             /* Bit number to be set */
```

```
/* Exit parameters
  None.
*/
unsigned byte_offset;
                               /* Byte offset into the bit vector */
if ((byte_offset = bitnum >> 3) > bv -> bv_bytes)
return 0; /* Bitnum is "off the end" of the vector */
                                                                                      ee
/* Set the appropriate bit in the vector. The byte offset
  has already been calculated. The bit number in the byte is calculated by AND ing the bit number with 0x07. The specified bit is then OR ed into the vector */
bv -> bv_bits[byte_offset] != (1 << (bitnum & 0x7));</pre>
                       /* Indicate completion */ .
return 1;
/* End of bv_set */
bv_test(bv,bitnum)
                               /* Test the specified bit number */
   /* This function returns a value that reflects the current
   setting of the specified bit. */
/* Entry parameters */
                              /* Pointer to bit vector */
/* Bit number to be set */
struct _bv *bv;
unsigned bitnum:
/* Exit parameters
  None.
×/
                                                                                      ff
                                /* Byte offset into the bit vector */
unsigned byte_offset;
if ((byte_offset = bitnum >> 3) > bv -> bv_bytes)
return 0; /* Bitnum is "off the end" of the vector */
/* Set the appropriate bit in the vector. The byte offset
  has already been calculated. The bit number in the byte
   is calculated by AND ing the bit number with 0x07.
The specified bit is then OR ed into the vector */
return bv -> bv_bits[byte_offset] & (1 << (bitnum & 0x7));</pre>
} /* End of by _tests */
/* Test bit vector nonzero */
bv_nz(bv)
/* This function tests each byte in the specified vector,
   and returns indicating whether any bits are set in
   the vector. */
/* Entry parameters */

hush by ¥hy: /* Pointer to bit vector */
/* Exit Parameters
   NZ = one or more bits are set in the vector 0 = all bits are off
                                                                                      gg
char *bits;
                        /* Pointer to bits in bit vector */
bits = bv -> bv_bits;
                                /* Set working pointer */
while (bits != bv -> bv_end) /* For entire bit vector */
        if (*bits++)
                                /* If nonzero */
                return bits--; /* Return pointer to NZ byte */
```

1

Figure 11-1. (Continued)

```
return O;
                           /* Indicate vector is zero */
                                                                             gg
} /* End of by_nz */
bv_and(bv3,bv1,bv2)
                            /* bv3 = bv1 & bv2 */
/* This function performs a boolean AND between the bytes
   of bit vector 1 and 2, storing the result in bit vector 3. */
/* Entry parameters */
struct _bv *bv1;
struct _bv *bv2;
                             /* Pointer to input bit vector */
                             /* Pointer to input bit vector */
/* Exit parameters */
struct _bv *bv3;
                            /* Pointer to output bit vector */
                                                                             hh
                            /* Working pointers to bit vectors */
char *bits1, *bits2, *bits3;
bits1 = bv1 -> bv_bits;
                            /* Initialize working pointers */
bits2 = bv2 -> bv_bits;
bits3 = bv3 -> bv_bits;
       /* AND ing will proceed until the end of any one of the bit
vectors is reached */
while (bits1 != bv1 -> bv_end &&
bits2 != bv2 -> bv_end &&
      bits3 != bv3 -> bv_end)
       £
              *bits3++ = *bits1++ & *bits2++; /* bv3 = bv1 & bv2 */
       3
} /* End of by_and */
bv or(bv3,bv1,bv2)
                            /* bv3 = bv1 or bv2 */
/* This function performs a boolean inclusive OR between the bytes
  of bit vectors 1 and 2, storing the result in bit vector 3. */
/* Entry parameters */
struct _bv *bv1;
struct _bv *bv2;
                            /* Pointer to input bit vector */
                            /* Pointer to input bit vector */
/* Exit parameters */
struct _bv *bv3;
                            /* Pointer to output bit vector */
                                                                             ii
char *bits1. *bits2. *bits3:
                            /* Working pointers to bit vectors */
bits1 = bv1 -> bv_bits;
                            /* Initialize working pointers */
bits2 = bv2 -> bv_bits;
bits3 = bv3 -> bv_bits;
       /* The OR ing will proceed until the end of any one of the bit
vectors is reached. */
while (bits1 != bv1 -> bv_end &&
bits2 != bv2 -> bv_end &&
      bits3 != bv3 -> bv_end)
       £
              *bits3++ = *bits1++ | *bits2++; /* bv3 = bv1 or bv2 */
       3
} /* End of by_or */
bv_disp(title,bv) /* Bit vector display */
/* This function displays the contents of the specified bit vector
  in hexadecimal. It is normally only used for debugging. */
                                                                             ij
/* Entry parameters */
                           /* Title for the display */
/* Pointer to the bit vector */
char *title;
struct _bv *bv;
```

```
/* Exit parameters
   None.
char *bits;
                                    /* Working pointer */
unsigned byte_count;
                                     /* Count used for formatting display */
/* Count for processing bits in a byte */
unsigned bit_count;
char byte_value;
                                               /* Value to be displayed */
printf("\nBit Vector : %s",title);
                                               /* Display title */
bits = by \rightarrow by bits:
                                               /* Set working pointer */
byte count = 0;
                                              /* Initialize count */
while (bits != bv -> bv_end)
                                               /* For the entire vector */
         if (byte_count % 5 == 0)
                                               /* Check if new line */
                                                                                                      jj
                                               /* Display bit number */
                   printf("\n%4d : ",byte_count << 3);</pre>
         byte_value = *bits++; /* Get the next byte from the vector */
         for (bit_count = 0; bit_count < 8; bit_count++)
                   /* Display the leftmost bit, then shift the value
                  left one bit */
if (bit_count == 4) putchar(' '); /* Separator */
putchar((byte_value & 0x80) ? '1' : '0');
byte_value <<= 1; /* Shift value left */</pre>
         }
                                                          /* Separator */
         byte_count++;
                           /* Update byte count */
} /* End of bv_disp */
/* End of LIBRARY.C */
```

Figure 11-1. (Continued)

Associated with the library of functions is another section of source code called "LIBRARY.H", shown in Figure 11-2. This "header" file must be included at the beginning of each program that calls any of the library functions.

For reasons of clarity, this chapter describes the simplest functions first, followed by the more complex, and finally by the utility programs that use the functions.

Several functions in the library and some definitions in the library header are not used by the utilities shown in this chapter. They have been included to illustrate techniques and because they might be useful in other utilities you could write.

```
#define LIBVN "1.0" /* Library version number */
/* This file contains groups of useful definitions.
    It should be included at the beginning of any program
    that uses the functions in LIBRARY.C */
/* Definition to make minor language modification to C. */
#define short char /* Short is not supported directly */
a
```

Figure 11-2. LIBRARY.H, code to be included at the beginning of any program that calls LIBRARY functions in Figure 11-1

```
/* One of the functions (bv_make) in the library uses the BDS C
function, alloc, to allocate memory. The following definitions
are provided for alloc. */
struct _header
                                          /* Header for block of memory allocated */
                                                                                                                   b
          struct _header *_ptr; /* Pointer to the next header in the chain */
unsigned _size; /* Number of bytes in the allocated block */
           3;
struct _header _base;
struct _header *_allocp;
                                          /* Declare the first header of the chain */
                                         /* Used by alloc() and free() functions */
/* BDOS function call numbers */
#define SETDISK 14
#define SEARCHF 17
                               /* Set (select) disk */
                              /* Search first */
#define SEARCHN 18
                              /* Search next */
#define DELETEF 19
                               /* Delete file */
                                                                                                                   С
#define GETDISK 25
                              /* Get default disk (currently logged in) */
                             /* Set DHA (Read/Write) Address */
/* Get disk parameter block address */
/* Get current user number */
/* Set current user number */
#define SETDMA 26
#define GETDPARM 31
#define GETUSER 32
#define SETUSER 32
/* Direct BIOS calls
    These definitions are for direct calls to the BIOS.
    WARNING: Using these makes program less transportable.
Each symbol is related to its corresponding jump in the
    BIOS jump vector.
    Only the more useful entries are defined. */
#define CONST
                               /* Console status *.
/* Console input */
#define CONIN
                     3
#define CONOUT 4
                              /* Console output */
#define LIST 5
#define AUXOUT 6
AUXIN 7
                              /* List output */
                              /* Auxiliary output */
                              /* Auxiliary input */
#define HOME
                               /* Home disk */
                     8
                                                                                                                   d
#define SELDSK 9
#define SETTRK 10
#define SETSEC 11
                               /* Select logical disk */
                               /* Set track */
/* Set sector */
                               /* Set DMA address */
#define SETDMA
                     12
#define DREAD
                     13
                               /* Disk read */
#define DWRITE 14
#define LISTST 15
                               /* Disk write */
                               /* List status */
                               /* Sector translate */
#define SECTRN 16
#define AUXIST 17
#define AUXOST 18
                               /* Auxiliary input status */
/* Auxiliary output status */
                               /* "Private" entries in jump vector */
                              /* Specific character I/O initialization */
/* Set watchdog timer */
#define CIOINIT 19
#define SETDOG 20
#define CBGADDR 21
                               /* Configuration block, get address */
/* Definitions for accessing the configuration block */
#define CB_GET 21
                                          /* BIOS jump number to access routine */
#define DEV_INIT 19
                                         /* BIOS jump to initialize device */
#define CB_DATE 0
                                         /* Date in ASCII *.
#define CB_TIMEA 1
#define CB_DTFLAGS 2
                                         /* Time in ASCII */
/* Date, time flags */
                                         /* This bit NZ means date has been set */
/* This bit NZ means time has been set */
#define TIME_SET 0x01
                                                                                                                  e
#define DATE_SET 0x02
#define CB_FIP 3
                                          /* Forced input pointer */
#define CB_SUM 4
                                          /* System start-up message */
                                          /* Console input */
#define CB CI 5
#define CB_CO 6
#define CB_AI 7
                                         /* Console output */
/* Auxiliary input */
#define CB_AO 8
                                          /* Auxiliary output */
```

Figure 11-2. (Continued)

```
#define CB_LI 9
                                         /* List input */
#define CB_LO 10
                                        /* List output */
#define CB_DTA 11
#define CB_C1224 12
                                        /* Device table addresses */
/* Clock 12/24 format flag */
                                        /* Real time clock tick rate (per second) */
#define CB_RTCTR 13
#define CB WDC 14
                                        /* Watchdog count */
#define CB_WDA 15
                                        /* Watchdog address */
#define CB_FKT 16
                                         /* Function key table */
#define CB_COET 17
                                        /* Console output escape table */ /
#define CB_DO_IS 18
                                         /* Device O initialization stream */
                                                                                                                   e
#define CB_D0_BRC 19
                                        /* Device O baud rate constant */
#define CB_D1_IS 20
#define CB_D1_BRC 21
                                         /* Device 1 initialization stream */
                                        /* Device 1 baud rate constant */
                                        /* Device 2 initialization stream */
/* Device 2 baud rate constant */
#define CB_D2_IS 22
#define CB D2 BRC 23
#define CB_IV 24
                                        /* Interrupt vector */
                                        /* Long term config. block offset */
#define CB_LTCBO 25
#define CB_LTCBL 26
                                        /* Long term config. block length */
#define CB_PUBF 27
                                        /* Public files flag */
#define CB_MCBUF 28
                                        /* Multi-command buffer */
#define CB_POLLC 29
                                        /* Polled console flag */
          /* Device numbers and names for physical devices */
/* NOTE: Change these definitions for your computer system */
#define T_DEVN 0
#define M_DEVN 1
#define P_DEVN 2
                                         /* Terminal */
                                                                                                                   f
                                         /* Modem */
                                        /* Printer */
#define MAXPDEV 2
                                         /* Maximum physical device number */
          /* Names for the physical devices */
                                                                                                                   g
#define PN_T "TERMINAL"
#define PN_M "MODEM"
#define PN P "PRINTER"
           /* Structure and definitions for function keys */
#define FK_ILENGTH 2
                                        /* No. of chars. input when func. key pressed
                                        NOTE: This does NOT include the ESCAPE. */
/* Length of string (not including fk_term) */
/* Number of function key entries in table */
#define FK_LENGTH 16
#define FK_LENGTH 16
#define FK_ENTRIES 18
                                                                                                                  h
struct _fkt
                                         /* Function key table */
          char fk_input[FK_ILENGTH];
                                                   /* Lead-in character is not in table */
          char fk_output[FK_LENGTH];
char fk_term;
                                                  /* Output character string */
/* Safety terminating character */
          3:
/* Definitions and structure for device tables */
           /* Protocol bits */
           /* Note: if the most significant bit is
set = 1, then the set_proto function
will logically OR in the value. This
               permits Input DTR to co-exist with
                                                                                                                   i
               XON or ETX protocol. */
#define DT_ODTR 0x8004
#define DT_OXON 0x0008
#define DT_OETX 0x0010
                                         /* Output DTR high to send (OR ed in) */
                                        /* Output DTR high t
/* Output XON */
/* Output ETX/ACK */
 #define DT_IRTS 0x8040
#define DT_IXON 0x0080
                                       /* Input RTS (OR-ed in) */
/* Input XON */
```

Figure 11-2. (Continued)

```
#define ALLPROTO OxDC
                                    /* All protocols combined */
                                    /* Device table */
struct _dt
                                    /* Filler */
         char dt_f1[14];
                                                                                                   i
         char dt_st1;
                                    /* Status byte 1 -- has protocol flags */
         char dt_st2;
                                    /* Status byte 2 */
                                    /* Filler */
         unsigned dt_f2;
         unsigned dt_etxml;
                                    /* ETX/ACK message length */
         char dt_f3[12];
                                    /* Filler */
         3 :
/* Values returned by the comp_fname (compare file name) */
#define NAME EQ 0
                          /* Names equal */
                                                                                                   i
#define NAME_LT 1
#define NAME_GT 2
                          /* Name less than mask */
                          /* Name greater than mask */
#define NAME NE 3
                          /* Name not equal (and comparison ambiguous) */
/* Structure for standard CP/M file control block */
#define FCBSIZE 36
                                    /* Define the overall length of an FCB */
struct _fcb
         £
                                   /* Logical disk (0 = default) */
         short fcb_disk;
         char fcb_fname[11];
                                   /* File name, type (with attributes) */
         short fcb_extent;
                                    /* Current extent */
                                                                                                   k
         unsigned fcb_s12;
                                    /* Reserved for CP/M */
         short fcb_recont;
                                    /* Record count used in current extent */
                                    /* Allocation blocks can be either */
         union
                                    /* Single or double bytes */
                  short fcbab_short[16];
                  unsigned fcbab_long[8];
                  } _fcbab;
         short fcb_currec;
                                    /* Current record within extent */
                                   /* Record for random read/write */
         char fcb_ranrec[3];
         3 ::
/* Parameter block used for calls to the directory management routines */
#define DIR_BSZ 128
                                   /* Directory buffer size */
struct _dirpb
         short dp_open;
                                   /* O to request directory to be opened */
                                   /* NZ when at end of directory */
/* NZ to write current sector to disk */
         short dp_end;
         short dp_write;
         struct _dir *dp_entry;
                                    /* Pointer to directory entry in buffer */
                                                                                                  1
                                           /* Directory sector buffer */
         char dp_buffer [DIR_BSZ];
         char dp_disk;
                                   /* Current logical disk */
         int dp_track;
                                   /* Start track */
         int dp_sector;
                                   /* Start sector */
                                  /* Number of directory entries */
         int dp_nument;
                                   /* Entries remaining to process */
         int dp_entrem;
                                  /* Number of sectors per track */
/* Number of allocation blocks per dir. entry */
/* Number of allocation blocks */
/* Allocation block size (in Kbytes) */
         int dp_sptrk;
         int dp_nabpde;
         unsigned dp_nab;
         int dp_absize;
         ):
/* The err_dir function is used to report errors found by the
   directory management routines, open_dir and rw_dir.
   Err_dir needs a parameter to define the operation being performed when the error occurred. The following definitions
                                                                                                   m
   represent the operations possible. */
                          /* Writing directory */
#define W_DIR 0
#define R_DIR
#define O DIR
                          /* Reading directory */
                 1
                ż
                          /* Opening directory */
```

Figure 11-2. (Continued)

```
/* Disk parameter block maintained by CPM */
struct _dpb
         unsigned dpb_sptrk;
                                      /* Sectors per track */
         short dpb_bshift;
                                     /* Block shift */
         short dpb_bmask;
                                      /* Block mask */
                                                                                                        n
                                     /* Extent mask */
         short dpb_emask;
         unsigned dpb_maxabn;
                                     /* Maximum allocation block number */
         unsigned dpb_maxden;
                                     /* Maximum directory entry number */
/* Allocation blocks reserved for */
         short dpb_rab0;
                                    /* directory blocks */
/* Disk changed workarea */
/* Track offset */
                                     /×
         short dpb_rab1;
unsigned dpb_diskca;
unsigned dpb_trkoff;
         3;
/* Disk directory entry format */
struct _dir f
         char de_userno;
                                    /* User number or 0xE5 if free entry */
                                     /* File name [8] and type [3] */
/* Extent number of this entry */
/* Number of 128-byte records used in last
         char de_fname[11];
         int de_extent;
         int de_recont;
                                                                                                        0
                                            allocation block */
                                      /* Allocation blocks can be either */
         union
                                            single or double bytes */
                                      /×
                   4
                   short de_short[16];
unsigned de_long[8];
                   } _dirab;
         3:
/* Disk request parameters for BIOS-level read/writes */
struct _drb
         short dr_disk;
unsigned dr_track;
                                     /* Logical disk A = 0, B = 1... */
/* Track (for SETTRK) */
                                                                                                        р
                                     /* Sector (for SETSEC) */
/* Buffer address (for SETDMA) */
         unsigned dr_sector;
         char *dr_buffer;
         3 .
/* Search control block used by directory scanning functions */
struct _scb
          ş
         short scb_userno;
char scb_fname[11];
short scb_extent;
char unused[19];
                                     /* User number(s) to match */
                                      /* File name and type */
                                      /* Extent number */
                                                                                                        q
                                      /* Dummy bytes to make this look like
                                            a file control block */
          short scb_length;
                                      /* Number of bytes to compare */
          short scb_disk;
                                      /* Current disk to be searched */
          unsigned scb_adisks;
                                      /* Bit map of disks to be searched.
                                            the rightmost bit is for disk A:. */
          3 :
/* Code table related definitions */
#define CT_SNF_0xFFFF /* String not found */
                                                                                                        r
struct _ct
                             /* Befine structure of code table */
          £
          unsigned _ct_code;
                                      /* Code value */
          char *_ct_sp;
                                     /* String pointer */
          3:
```

Figure 11-2. (Continued)

Figure 11-2. (Continued)

Library Functions

This section describes the library functions and the sections from the header file that must be included at the beginning of each utility program.

A Minor Change to C Language

One minor problem with the BDS C Compiler is that it does not support "short" integers, or integers that are only a single byte long. It is convenient to declare certain values as short to serve as a reminder of the standard type definition. Therefore, the BDS C compiler must be "fooled" by declaring these values to be single characters. To do this, the library header file contains the declaration

#define short char.

shown in Figure 11-2, section a.

The "#define" tells the first part of the C compiler, the preprocessor, to substitute the string "char" (which declares a character variable) whenever it encounters the string "short" (which would ordinarily declare a short integer in standard C).

Note that character strings enclosed in "/*" and "*/" are regarded as comments and are ignored by the compiler.

BDOS Calls

The standard library of functions that comes with the BDS C compiler includes a function to make BDOS calls, called "bdos." It takes two parameters, and a typical call is of the following form:

bdos(c.de);

The "c" parameter represents the value that will be placed into the C register. This is the BDOS function code number. The "de" is the value that will be placed in the DE register pair.

396 The CP/M Programmer's Handbook

The library header contains definitions (#define declarations) for BDOS functions 14 through 32, making these functions easier to use (Figure 11-2, c). Function 32 (Get/Set Current User Number) has two definitions; the "de" parameter is used to differentiate whether a get or a set function is to be performed.

BIOS Calls

The BDS C standard library also contains two functions that make direct BIOS calls. These are "bios" and "biosh." They differ only in that the bios function returns the value in the A register on return from the BIOS routine, whereas biosh, as its name implies, returns the value in the HL register pair. Examples of their use are

bios(jump_number,bc);

and

biosh(jump_number,bc,de);

Both functions take as their first parameter the number of the jump instruction in the BIOS jump vector to which control is to be transferred. For example, the console-status entry point is the third JMP in the vector. Numbering from 0, this would be jump number 2.

The library header file contains #defines for BIOS jumps 2 through 21 (Figure 11-2, d). The last group of these #defines (19 through 21) is for the "private" additions to the standard BIOS jump vectors described in Chapter 8.

Remember, though, that using direct BIOS calls makes programs more difficult to move from one system to another.

BIOS Configuration Block Access

As you may recall, the configuration block is a collection of data structures in the BIOS. These structures are used either to store the current settings of certain user-selectable options, or to point to other important data structures in the BIOS.

One of the "private" jumps appended to the standard BIOS jump vector transfers control to a routine that returns the address in memory of a specified data structure. For example, if a utility program needs to locate the word in the BIOS that determines from which physical device the console input is to read, it can transfer control to jump 21 in the BIOS jump vector (actually the 22nd jump) with a code value of 5 in the C register. This jump transfers control to the CB\$Get\$-Address code, which on its return will set HL to the address of the console input redirection vector. The utility program can then read from or write into this variable. The library header file contains #define declarations relating the code values to mnemonic names (Figure 11-2, e).

You will need to refer to the source code in Figure 8-10 to determine whether the address returned by the BIOS function is the address of the data element or the

address of a higher-level table that in turn points to the data element.

In order to access the current system date, for example, you would include the following code:

The ptr_to_date can then be used to access the date directly.

During initial debugging of a utility, it is useful to be able to intercept all such accesses to the configuration block, partly to reassure yourself that the utility program is working as it should, and partly to ensure that the BIOS routine is returning the correct addresses to the data structures. Therefore, the utility library contains a function, "get_cba," that gets a configuration block address (Figure 11-1, a).

At first, it appears that get_cba is declared as a function that returns a pointer to characters. This is not strictly true. Sometimes the address it returns will point to characters, sometimes to integers, and sometimes to structures (such as the function key table).

The "printf" instruction has been left in the function in anticipation of debugging a utility. If you need to see some debug output whenever the get_cba function is used, delete the "/*" and "*/" surrounding the "printf" and recompile the library.

BIOS Function Key Table Access

The BIOS shown in Figure 8-10 contains code to recognize when an incoming escape sequence indicates that one of the terminal's function keys has been pressed. Instead of returning just the escape sequence, the console driver injects a previously programmed string of characters into the console input stream. For example, on a DEC VT-100 terminal, when the PF1 function key is pressed, the terminal emits the following character sequence: ESCAPE, "O", "P". The function key table contains the "OP" and a 00H-byte-terminated string of characters to be injected into the console input stream. In Figure 8-10, the example string is "FUNCTION KEY 1", LINE FEED. The library header file contains a declaration for the structure of the function key table (Figure 11-2, h).

Note the use of "#define" to declare the length of the incoming characters emitted by the terminal as well as the length of the output string.

In order to access a function key table entry, you must declare a pointer to a "_fkt" structure like this:

The get_cba function is used to return the address of the first entry in the function key table and set a pointer to it. Then the printf function (part of the

standard BDS C library) is used to print out the first string, which gets substituted for the "%s" in the quoted string. Note that the statement

++ptr_to_fkt

does not just add one to the pointer to the function key table—it adds whatever it takes to move the pointer to the next *entry* in the table.

BIOS Device Table Access

The device tables are important structures for the serial devices served by the console, auxiliary, and list device drivers in the BIOS. They are declared at line 1500 in Figure 8-10.

The get_cba function does not return a pointer to a specific device table, but a pointer to a table of device table addresses. Each entry in the address table corresponds to a specific device number. If there is no device table for a specific device number, then the corresponding entry in the table will be set to zero. the library header file contains definitions for the device table (Figure 11-2, i).

The device tables contain, among other things, the current serial line protocols used to synchronize the transmission and reception of data by the device drivers and the physical devices. An example utility, PROTOCOL, is shown later in the chapter. The example #define declarations and structure definition shown here are modeled on the requirements of this utility. The only relevant bytes are the two status bytes dt_st1 and dt_st2 and the message length used with the ETX/ACK protocol, dt_etxml. The #defines shown are for the specific bits in the device table's status bytes. The PROTOCOL utility uses the most significant bit to indicate whether a given protocol setting can coexist with others.

To access these fields, use the following code:

```
struct _ppdt
     £
     char *pdt[16];
                         /* Array of 16 pointers to device tables */
                         /* Pointer to array of 16 pointers */
     } *ppdt;
                         /* Pointer to device table */
struct dt *dt;
ppdt = get_cba(CB_DTA);
                              /* Set pointer to array of pointers */
dt = ppdt -> pdt[device_no]; /* Set pointer to specified device
                                 table */
if (Idt)
     printf("\nError - no device table for this device.");
dt -> dt_etxml = 0;
                              /* Clear ETX message length */
```

BIOS Disk Parameter Block Access

Several of the utility programs shown in this chapter must access the file directory on a given logical disk. The disk parameter block (DPB) indicates the size and location of the file directory. The library header contains a structure definition that describes the DPB (Figure 11-2, n).

To locate the DPB, you can make a direct BIOS call to the SELDSK routine, which returns the address of the disk parameter header (DPH). You then can access the DPB pointer in the DPH. Alternatively, using the BDOS, you can make the required disk the default disk and then request the address of its DPB. The code for the latter method is shown in the get_dpb function included in the utility library (Figure 11-1, u).

The get_dpb function uses a BIOS SELDSK function first to see if the specified disk is legitimate. Only then does it use the BDOS.

Reading or Writing a Disk Using the BIOS

When you write a program that uses direct BIOS calls, you increase the possibility of problems in moving the program from one system to another. However, in certain circumstances it is necessary to use the BIOS. Reading and writing the file directory is one of these; the BDOS cannot be used to access the directory directly. The library header contains a structure declaration for a parameter block that contains the details of an "absolute" disk read or write (Figure 11-2, p).

Note the pointer to the 128-byte data buffer used to hold one of CP/M's "records."

The disk read and write functions are rd_disk (Figure 11-1, k) and wrt_disk (Figure 11-1, l). Both of them take a _drb as an input parameter, and both call the set_disk function to make the individual BIOS calls to SELDSK, SETTRK, and SETSEC.

Of special note is the code in set_disk (Figure 11-1, m) that converts a logical sector into a physical sector using the sector translation table and the SECTRAN entry point in the BIOS.

File Directory Entry Access

All of the utility programs that access a disk directory share the same basic logic regardless of their specific task. This logic can be described best in pseudo-code:

```
while (not at the end of the directory)
{
    access the next directory entry
    if (this entry matches the current search criteria)
    {
        process the entry
    }
}
```

There are two ways of implementing this logic. The first uses the BIOS to read the directory. Entries are presented to the utility exactly as they occur in the file directory. The second uses the BDOS functions Search First and Search Next and accesses the directory file-by-file rather than by entry. This latter method is more suited to utilities that process files rather than entries. The ERASE utility, described later in this chapter, illustrates this second method.

Three groups of functions are provided in the library: to access the next entry in the directory, to match the name in the current entry against a search key, and to assist with processing the directory.

Directory Accessing Functions

A number of functions involve access to the file directory. The first group of such functions performs the following:

get_nde (get next directory entry; Figure 11-1, n)

This function returns a pointer to the next directory entry, or returns zero if the end of the directory has been reached.

open_dir (open directory; Figure 11-1, o)

This function is called by get_nde to open up a directory for processing.

rw_dir (read/write directory; Figure 11-1, p) This function reads or writes the current directory sector.

err_dir (error on directory; Figure 11-1, q) This general-purpose routine displays an error message if the BIOS indicates that it had problems either reading or writing the directory.

All of these functions use a directory parameter block to coordinate their activity. The library header contains the definitions for this structure (Figure 11-2, 1), as well as #define declarations for operation codes used by the directory-accessing functions (Figure 11-2, m).

Before calling get_nde, the calling program needs to set dp_open to zero (forcing a call to open_dir) and the dp_disk field to the correct logical disk. The open_dir function sets up all of the remaining fields, using get_dpb to access the disk parameter block for the disk specified in dp_disk.

Of the remaining flags, dp_end will be set to true, when the end of the directory is reached, and dp_write must be nonzero for rw_dir to write the current sector back onto the disk.

The get_nde function includes all of the necessary logic to move from one directory entry to the next, reading in the next sector when necessary, and writing out the previous sector if the dp_write flag has been set to a nonzero value by the calling program. It also counts down on the number of directory entries processed, detecting and indicating the end of the directory.

The code at the beginning of the function calls open_dir if the dp_open flag is false. Note the code at the end of open_dir that sets the number of allocation blocks per directory entry (dp_nabpde). This number is computed from the maximum

allocation block number in the disk parameter block. If it is larger than 255, each allocation block must occupy a word, and there will be eight blocks per directory entry. If there are 255 or fewer allocation blocks, each will be one byte long and there will be 16 per entry. The allocation block size, in K bytes, is computed from a simple formula.

In the early stages of debugging utilities, comment out the line that makes the call to wrt_disk. This will prevent the directory from being overwritten. You then can test even those utilities that attempt to erase entries from the directory without any risk of damaging any data on the disk.

The last function in this group, err_dir, is a common error handling function for taking care of errors while reading or writing the directory.

Directory Matching Functions

The second group of functions that access the file directory matches each directory entry against specific search criteria. These include the following functions:

setscb (set search control block; Figure 11-1, r)

A search control block (SCB) is a structure that defines the entries in the directory that are to be selected for processing.

comp_fname (compare file name; Figure 11-1, f)

This function compares the file name in the current directory entry with the one specified in the search control block.

The library header contains the structure definition for the search control block (Figure 11-2, q). This SCB is a hybrid structure. The first part of it is a cross between a file control block (FCB) and a directory entry. The last three fields, scb_length, scb_disk, and scb_adisks, are peculiar to the search control block. Note that its overall length is the same as an FCB's so that the standard BDS C function set_fcb can be used. This function sets the file name and type into an FCB, replacing "*" with as many "?" characters as are required, and clears all unused bytes to zero.

The scb_length field indicates to the comp_fname (compare file name) function how many bytes of the structure are to be compared. This field will be set to 12 to compare the user number, file name, and type, or to 13 to include the extent number.

Note that scb_disk is the *current* disk to be searched, whereas scb_adisks is a bit map with a 1 bit corresponding to each of the 16 possible logical disks that must be searched.

The search control block is initialized by the setscb function.

Note the form of the file name that setscb expects to receive. This is described in the comments at the beginning of the function.

Several of the utility programs use their own special versions of setscb,

renaming it ssetscb (special setscb) to avoid the library version being linked into the programs.

The complementary function comp_fname is used to compare the first few bytes of the current directory entry to the corresponding bytes of the SCB.

The comp_fname function performs a specialized string match of the user number, the file name, the file type, and, optionally, the extent number. A "?" character in the search control block file name, type, and extent will match with any character in the file directory entry. However, in the SCB user number, a "?" will only match a number in the range 0 to 15; it will not match a directory entry that has the user number byte set to E5H (or 0xE5, as hexadecimal notation in C).

This function also returns one of several values to indicate the result of the comparison. These values are defined in the library header file (Figure 11-2, j).

Directory Processing Functions

The final group of functions that access the directory are those that help process the directory entries themselves. These functions use a structure definition to access each directory entry (Figure 11-2, 0).

A union statement is used for the allocation block numbers. These can be single- or two-byte entries, depending on the maximum number of allocation blocks that must be represented. The union statement tells the BDS C compiler whether there will be a 16-byte array of short integers (characters) or an array of eight unsigned two-byte integers.

The functions contained in this group can be divided into three subgroups:

- Those that deal with converting directory entries for display on the console.
- Those that deal with a "disk map"—a convenient array for representing logical disks and the user numbers they contain.
- Those that deal with "bit vectors"—a convenient representation of which allocation blocks on a logical disk are in use or available.

The library contains only one function to convert a directory-entry file name into a suitable form for display on the console. This is the conv_dfname function (Figure 11-1, h). It takes the information from the specified directory entry (or, as a convenience, a search control block) and formats it into a string of the form

uu/d:filename.typ

The "uu" specifies the user number and the "d" specifies the disk identification.

The repetitive code at the end of the function is necessary to make sure that the characters in the file type do not have their high-order bits set. These bits are the file attributes. If they are set, they can render the characters nondisplayable on some terminals.

The second subgroup of functions, those that manipulate a "disk map," produce an array that looks like this:

```
Disks

'

v User Numbers -->

A 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Used Free

B

:

P
```

This disk map is used by several utility programs. For example, the SPACE utility displays a disk map that shows, for each logical disk in the system, and for each user on each logical disk, how many K bytes of disk space are in use. The totals at the right show the total of used and free space. In another example, the FIND utility shows how many files on each disk and in each user number match the search name.

Each utility program that uses a disk map is coded:

unsigned disk_map[16][18];

Two functions are provided in the library to deal with the disk map:

```
dm_clr (disk map clear; Figure 11-1, s)
```

This function fills the entire disk map with zeros.

dm_disp (disk map display; Figure 11-1, t)

This function displays the horizontal and vertical caption lines for the disk map and then converts each element of the disk map to a decimal number.

The first function, dm_clr, uses one of the standard BDS C functions to set a block of memory to a specific value. It presumes that the disk map is 16×18 elements, each two bytes long.

The second function, dm_disp, prints horizontal lines only for those disks specified in the bit map parameter. Here is an example of its output:

```
0
          1
              2
                  з
                      4
                                10 11 12 13 14 15 Used Free
      1
A:
        1
                                                           15 241
             74
R:
     66 20
                 50
                      З
                                                          245 779
C:
     -- None --
                                                            0 1024
(NOTE: All user groups would be shown on the terminal.)
```

The final subgroup deals with processing "bit vectors." A bit vector is a string of bits packed eight bits per byte. Each bit is addressed by its relative number along the vector; the first bit is number 0.

An example of why bit vectors are used is a utility program that needs to scan the directory of a disk and build a structure showing which allocation blocks are in use. It can do this by accessing each active directory element and, for each nonzero allocation block number, setting the corresponding bit number in a bit vector.

The library header has a structure definition for a bit vector (Figure 11-2, s).

This vector contains the overall length of the bit vector in bytes, and two pointers. The first points to the start of the vector, the second to the end. The bytes that contain the vector bits themselves are allocated by the alloc function—one of the standard BDS C functions.

The following bit vector functions are provided in the library:

bv_make (bit vector make; Figure 11-1, cc)

This function allocates memory for the bit vector (using the standard mechanism provided by BDS C) and sets all of the bits to zero.

bv_fill (bit vector fill; Figure 11-1, dd)

This fills a specified vector, setting each byte to a specified value.

bv_set (bit vector set; Figure 11-1, ee)

This sets the specified bit of a vector to one.

bv_test (bit vector test; Figure 11-1, ff)

This function returns a value of zero or one, reflecting the setting of the specified bit in a bit vector.

bv_nz (bit vector nonzero; Figure 11-1, gg)

This returns zero or a nonzero value to reflect whether *any* bits are set in the specified bit vector.

bv_and (bit vector AND; Figure 11-1, hh)

This function performs a Boolean AND between two bit vectors and places the result into a third vector.

bv_or (bit vector OR; Figure 11-1, ii) This is similar to bv_and, except that it performs an inclusive OR on the two input vectors.

bv_disp (bit vector display; Figure 11-1, jj)

This function displays a caption line and then prints out the contents of the specified bit vector as a series of zeros and ones. Each byte is formatted to make the output easier to read.

The by_make function uses the alloc function to allocate a block from the unused part of memory between the end of a program and the base of the BDOS. It requires that two data structures be declared at the beginning of the program. These structures are declared in the library header file (Figure 11-2, b).

The by_fill function uses the standard BDS C setmem function.

The bv_set function converts the bit number into a byte offset by shifting the bit number right three places. The least significant three bits of the original bit number specify which bit in the appropriate byte needs to be ORed in.

The bv_test function is effectively the reverse of bv_set. It accesses the specified bit and returns its value to the calling program.

The by_nz function scans the entire bit vector looking for the first nonzero

byte. If the entire vector is zero, it returns a value of zero. Otherwise, it returns a pointer to the first nonzero byte.

Both bv_and and bv_or functions take three bit vectors as parameters. The first vector is used to hold the result of either ANDing or ORing the second and third vectors together. Both of these functions assume that the output vector has already been created using bv_make. The shortest of the three vectors will terminate the bv_and or bv_or function; that is, these functions will terminate when they reach the end of the first (shortest) vector.

The final function, bv_disp, displays the title line specified by the calling program, and then displays all of the bits in the vector, with the bit number of the first bit on each line shown on the left.

None of the utility programs uses by_disp—it has been left in the library purely as an aid to debugging.

Here is an example of bv_disp's output:

```
Bit Vector : Allocation Blocks in Use
  0 : 0000 0000 0001 1000 1000 0001
                                        1111 1111
                                                   1111 1111
  40 : 1111 1111
                  1111 1111
                             1111 1111
                                        1110 1011
                                                   0000 0000
 80 : 1100 0000
                 1111 1100
                            1111 1001
                                        1100 0000
                                                   1001 1111
 120 : 1110 1100 0001 1111 0000 0000
                                                   0001 1110
                                        1101 1000
 160 : 1111 1111
                  1110 1111 1110 1111
                                        0000 0111
                                                   0000 0111
 200 : 1111 0010
```

Checking User-Specified Parameters

The C language provides a mechanism for accessing the parameters specified in the "command tail." It provides a count of the number of parameters entered, "argc" (argument count), and an array of pointers to each of the character strings, "argv" (argument vector). At the beginning of the main function of each program you must define these two variables like this:

```
main(argc,argv)
{
  int argc;  /* Argument count */
  char *argv[];  /* Array of pointers to char. strings */
  :
  /* Remainder of main function */
  ;
}
```

Consider the minimum case—a command line with just the program name on it:

A>command

The convention is that the first argument on the line is the name of the program itself. Hence argc would be set to one, and argv[0] would be a pointer to the program name, "command."

Next consider a more complex case — a command line with parameters like the following:

```
A>command param1 123
```

In this case, argc will be three; argv[1] will be a pointer to param1; and argv[1][0] will access the 0 (the first) character of argv[1]—in this case the character "p."

To detect whether the second parameter is present and numeric, the code will be

In most of the utilities, you will get a much "friendlier" program if the user need only specify enough characters of a parameter to distinguish the value entered from the other possible values. For example, consider a program that can have as a parameter one of the following values: 300, 600, 1200, 2400, 4800, 9600, or 19200. It would be convenient if the user needed to type only the first digit, rather than having to enter redundant keystrokes. However, the values 1200 and 19200 would then be ambiguous. The user would have to enter 12 or 19. Novice users often prefer to specify the entire parameter for clarity and security.

The standard C library provides a character string comparison function, strcmp. Unfortunately, this function does not provide for the partial matching just described. Therefore, the library includes two special functions that do make this possible: sstrcmp (substring compare, Figure 11-1, d) and usstrcmp (uppercase substring compare, Figure 11-1, e). The latter function is necessary when you need to compare a substring that could contain lowercase characters; it converts characters to uppercase before the comparison.

To assist with character string manipulation, two additional functions have been included in the library. These are strscn (string scan, Figure 11-1, b) and ustrcmp (uppercase string compare, Figure 11-1, c).

Using Code Tables

A code table is a simple structure used by all of the utility programs that accept parameters that can have any of several values. The library header contains a structure definition for a code table (Figure 11-2, r).

A code table entry contains an unsigned code value and a pointer to a character string. It is used in the utility programs wherever there is a need to relate some arbitrary code number or bit pattern to an ASCII character string. For example, to program a serial port baud-rate-generator chip to various baud rates requires different time constants for each rate. Users do not need to know what these numbers are; they only need to be able to specify the baud rate as an ASCII string. Thus, a code table is set up as follows:

Baud Rate Constant	User's Name
0x35	"300"
0x36	"600"
0x37	"1200"
0x3A	"2400"
0x3C	"4800"
0x3E	"9600"
0x3F	"19200"

A utility program now needs to be able to perform various operations using the code table:

- Given the input parameter on the command tail, the utility must check whether the ASCII string is in the code table, display all of the legal options on the console if it is not, and return the code value for subsequent processing if it is.
- Given the current baud rate constant (held in the BIOS), the utility must scan the code table and display the corresponding ASCII string to tell the user the current baud rate setting.

The library includes specialized functions to do this, plus some additional functions to make code tables more generally usable. These functions are

ct_init (code table initialize; Figure 11-1, v)

This function initializes a specific entry in a code table, setting the code value and the pointer to the character string.

ct_parc (code table parameter return code; Figure 11-1, w)

This performs an uppercase substring match on the specified key string, returning either an error (the value CT_SNF—string not found) or a code value.

ct_code (code table return code; Figure 11-1, x)

This function is similar to ct_parc in that it scans a code table and returns the corresponding code. It differs in the way that the comparison is done. The entire search string is compared with the string in the code table entry. A match only occurs when all characters are the same.

ct_disps (code table display strings; Figure 11-1, y)

This function displays all strings in a given code table. It is used either when the user has entered an invalid string, or when the utility program is requested to show what options are available for a parameter.

ct_index (code table return index; Figure 11-1, z)

This function, given a string, searches the code table and returns the index

of the entry that has a string matching the search string. The index is not the code value; it is the number of the entry in the table.

ct_stri (code table string index; Figure 11-1, aa)

This function, given an entry index number, returns a pointer to the string in that entry.

ct_strc (code table string code; Figure 11-1, bb)

This function, given a code number, returns a pointer to the string in the entry that has a matching code number.

Accessing a Directory via the BDOS

One problem associated with accessing the file directory directly, as illustrated by earlier functions, is that the program is presented with directory entries in exactly the order that they occur in the directory. For some programs, such as those that process groups of files, it is better to use the BDOS Search First and Search Next functions to access the directory.

Using the BDOS, the program can process the first file name to match an ambiguous search key, then go back to the BDOS to get the name of the next file, and so on. The library header contains a structure definition for a standard CP/M file control block (Figure 11-2, k).

Notice that the first byte of the FCB is a disk number rather than the user number of the directory entry. Note also the use of a union statement to describe the allocation block numbers.

The standard BDS C library contains a function, setfcb, that is given the address of an FCB and a pointer to a string containing a file name. It converts any "*" in the name to the appropriate number of "?", and fills the remainder of the FCB with zeros.

The example library contains the following functions designed for BDOS file directory access:

get_nfn (get next file name; Figure 11-1, i)

This function is given a pointer to an ambiguous file name and a pointer to an FCB. It returns with the FCB set up to access the next file that matches the ambiguous file name.

srch_file (search for file; Figure 11-1, j)

This function, used by get_nfn, issues either a Search First or a Search Next BDOS call.

conv_fname (convert file name; Figure 11-1, g)

This function converts a file name from an FCB into a form suitable for display on the console. It is similar to the conv_dfname function described earlier except that it outputs only the disk, file name, and type (not the user number) in the form

d:filename.typ

To signal the get_nfn function that you want the first file name, you must set the most significant bit of the first byte, the disk number.

Here is an example showing how to use the get_nfn function:

```
struct _fcb fcb;
                          /* Declare a file control block */
setmem(fcb,FCB SIZE,0);
                         /* Clear FCB to zeros */
fcb.fcb_disk = 0x80;
                         /* Mark FCB for "first time" */
while (get_nfn(fcb,"B:XYZ*.*"))
                         /* Until get_nfn returns a zero */
     £
                         /* Open the file using FCB */
     while
                         (/* Not at end of file */)
          £
                          /* Process next record or
                               Character in file*/
          3
                         /* Close the file */
     ł
```

The quoted string "B:XYZ*.*" could also be just a pointer to a string, or a parameter on the command line, argv[n].

The last function for BDOS processing of the file directory, conv_fname, is used to convert a file name for output to a terminal. Again, the repetitive code at the end clears the file attribute bits to avoid any side effects from the terminal.

Utility Programs Enhancing Standard CP/M

This group of utilities is designed to enhance those supplied by Digital Research. They do not take advantage of any special features of the enhanced BIOS in Figure 8-10 and can be used on *any* CP/M Version 2.2 installation.

With the exception of the ERASE utility, all of the utilities scan down the file directory using BIOS calls, as described earlier in this chapter.

ERASE — A Safer Way to Erase Files

There are two disadvantages to the Console Command Processor's built-in ERA command. First, it will unquestioningly erase groups of files. Second, if you have a file name with nongraphic or lowercase characters, you cannot use the ERA command, as the CCP converts the command tail characters to uppercase and terminates a file name on encountering any strange character in the string.

The ERASE utility shown in Figure 11-3 erases groups of files, but it asks the user for confirmation before it erases each file.

Rather than use the BIOS to access each directory entry, it uses the get_nfn function, which then calls the BDOS. Thus ERASE functions equally well for files

that have multiple entries in the directory. It can use the BDOS Delete File function to erase all extents of a given file.

Here is an example console dialog showing ERASE in operation:

```
P3A>erase<CR>
ERASE Version 1.0 02/23/83 (Library 1.0)
Usage :
        ERASE {d:}file_name.typ
P3A>erase *.com<CR>
ERASE Version 1.0 02/23/83 (Library 1.0)
Searching for file(s) matching A:??????.COM.
        Erase A: UNERASE .COM y/n? n
        Erase A:TEMP1 .COM y/n? y <== Will be Erased!
        Erase A:TEMP2 .COM y/n? n
        Erase A:TEMP3 .COM y/n? n
        Erase A:TEMP4 .COM y/n? \overline{y} <== Will be Erased!
        Erase A: ERASE .COM y/n? n
Erasing files now...
                       .COM erased.
        File A:TEMP1
        File A:TEMP4
                       .COM erased.
```

```
#define VN "1.0 02/24/83"
/* FRASE
   This utility erases the specified file(s) logically
   by using a BDOS delete function. */
#include <LIBRARY.H>
struct _fcb amb_fcb;
struct _fcb fcb;
                                 /* Ambiguous name file control block */
/* Used for BDOS search functions */
char file_name[20];
                                   /* Formatted for display: d:FILENAME.TYP */
                                    /* Current logical disk at start of program */
/* ERASE saves the FCB's of the all the
files that need to be erased in the
short cur_disk;
                                        following array */
#define MAXERA 1024
struct _fcb era_fcb[MAXERA];
                                     /* Count of number of files to be erased */
int ecount:
int count:
                                     /* Used to access era_fcb during erasing */
main(argc,argv)
                           /* Argument count */
short arge;
char *argv[];
                           /* Argument vector (pointer to an array of char. */
.
printf("\nERASE Version %s (Library %s)",VN,LIBVN);
pht use(argo): /* Check usage */
cur_disk = bdos(GETDISK);
                                    /* Get current default disk */
ecount = 0;
                                    /* Initialize count of files to erase */
setfcb(amb_fcb,argv[1]);
                                    /* Set ambiguous file name */
if (amb_fcb.fcb_disk)
                                    /* Check if default disk to be used */
         bdos(SETDISK,amb_fcb.fcb_disk + 1);
                                                       /* Set to specified disk */
```

Figure 11-3. ERASE.C, a utility that requests confirmation before erasing

```
/* Convert ambiguous file name for output */
conv_fname(amb_fcb,file_name);
printf("\n\nSearching for file(s) matching %s.",file_name);
        /* Set the file control block to indicate a "first" search */
fcb.fcb_disk != 0x80; /* OR in the ms bit */
        /* While not at the end of the directory, set the FCB
           to the next name that matches */
while(get_nfn(amb_fcb,fcb))
        conv_fname(fcb,file_name);
        /* Ask whether to erase file or not */
printf("\n\tErase %s y/n? ",file_name);
if (toupper(getchar()) == 'Y')
                printf(" <== Will be erased!");</pre>
                /* add current fcb to array of FCB's */
movmem(fcb,&era_fcb[ecount++],FCBSIZE);
                         /* Check that the table is not full */
                 if (ecount == MAXERA)
                         £
                         printf("\nWarning : Internal table now full. No more files can be erased");
printf("\n until those already specified have been erased.");
                         break; /* Break out of while loop */
        3
                 /* All directory entries processed */
if (ecount)
        printf("\n\nErasing files now...");
        for (count = 0:
     count < ecount;
     count++)
        £
        /* error? */
        else
                         /* File erased */
                printf("\n\tFile %s erased.",file_name);
bdos(SETDISK,cur_disk); /* reset to current disk */
chk_use(argc)
                         /* Check usage */
/* This function checks that the correct number of
  parameters has been specified, outputting instructions if not. */
/* Entry parameter */
               /* Count of the number of arguments on the command line */
int arge:
ŧ
        /* The minimum value of argc is 1 (for the program name itself),
            so argc is always one greater than the number of parameters
           on the command line */
if (argc != 2)
        ş
        printf("\nUsage :");
        printf("\n\tERASE {d:}file_name.typ");
        exit():
3
```

Figure 11-3. (Continued)

UNERASE — Restore Erased Files

UNERASE, as its name implies, can be used to "revive" an accidentally erased file. Only files whose allocation blocks have not been reallocated to other files can be revived. The UNERASE utility shown in Figure 11-4 builds a bit vector of all the allocation blocks used by active directory entries. Then it builds a bit vector for all the allocation blocks required by the file to be UNERASEd. If a Boolean AND between the two vectors yields a nonzero vector, then one or more blocks that originally belonged to the erased file are now allocated to other files on the disk.

```
#define VN "1.0 02/12/83"
/* UNERASE --
    This utility does the inverse of ERASE: it restores
    specified files to the directory by changing the first byte of
    their directory entries from 0xE5 back to the specified user
    number, */
#include <LIBRARY.H>
struct _dirpb dir_pb;
                                       /* Directory management parameter block */
struct _dir *dir_entry;
struct _scb scb;
struct _scb scba;
                                        /* Pointer to directory entry */
/* Search control block */
                                         /* SCB set up to match all files */
                                        /* CP/M's disk parameter block */
/* CP/M's disk parameter block */
/* Bit vector for blocks in use */
/* Bit vector for file to be unerased */
/* Bit vector for those extents unerased */
struct _dpb dpb;
struct _bv inuse_bv;
struct _bv file_bv;
struct _bv extents;
char file_name[20];
                                         /* Formatted for display : un/d:FILENAME.TYP */
short cur_disk;
                                        /* Current logical disk at start of program
                                              NZ = show map of number of files */
int count;
                                         /* Used to access the allocation block numbers
                                               in each directory entry */
                                         /* User in which the file is to be revived */
int user:
main(argc.argv)
                              /* Argument count */
short arge:
char *argv[];
                              /* Argument vector (pointer to an array of chars.) */
printf("\nUNERASE Version %s (Library %s)",VN,LIBVN);
chk use(argc);
                                         /* Check usage */
cur_disk = bdos(GETDISK);
                                         /* Get current default disk */
          /* Using a special version of the set search-control-block utility,
              set the disk, name, type (no ambiguous names), the user number
to match only erased entries, and the length to compare
              the user, name, and type.
This special version also returns the disk_id taken from
the file name on the command line. */
if ((dir_pb.dp_disk = ssetscb(scb,argv[1],0xE5,12)) == 0)
          { /* Use default disk */
dir_pb.dp_disk = cur_disk;
else
                    /* make disk A = 0, B = 1 (for SELDSK) */
          dir_pb.dp_disk--;
printf("\nSearching disk %d.",dir_pb.dp_disk);
if(strscn(scb."?"))
                              /* Check if ambiguous name */
          printf("\nError -- UNERASE can only revive a single file at a time.");
          exit():
```

Figure 11-4. UNERASE.C, a utility program that "revives" erased files

```
/* Set up a special search control block that will match with
             all existing files. */
ssetscb(scba."*.*".'?'.12):
                                       /* Set file name and initialize SCB */
if (argc == 2)
                                       /* No user number specified */
          user = bdos(GETUSER,0xFF);
                                               /* Get current user number */
else
ş
          user = atoi(argv[2]);
                                                /* Get specified number */
          if (user > 15)
                   ş
                   printf("\nUser number can only be 0 - 15.");
                   exit();
3
/* Build a bit vector that shows the allocation blocks
currently in use. SCBA has been set up to match all
active directory entries on the disk. */
build_bv(inuse_bv,scba);
/* Build a bit vector for the file to be restored showing
    which allocation blocks will be needed for the file. */
if (!build_bv(file_bv,scb))
          printf("\nNo directory entries found for file %s.",
                   argv[1]);
          exit();
          3
 /* Perform a boolean AND of the two bit vectors. */
bv_and(file_bv,inuse_bv,file_bv);
/* Check if the result is nonzero -- if so, then one or more
of the allocation blocks required by the erased file is
already in use for an existing file and the file cannot
    be restored. */
if (bv_nz(file_bv))
          printf("\n--- This file cannot be restored as some parts of it");
          printf("\n
                          have been re-used for other files! ---");
          exit():
3
/* Continue on to restore the file by changing all the entries
    in the directory to have the specified user number.
    Note: There may be several entries in the directory for
    the same file name and type, and even with the same extent
number. For this reason, a bit map is kept of the extent
    numbers unerased -- duplicate extent numbers will not be
    unerased. */
/* Set up the bit vector for up to 127 unerased extents */
bv make(extents,16);
                                       /* 16 * 8 bits */
/* Set the directory to "closed", and force the get_nde
   function to open it. */
dir_pb.dp_open = 0;
/* While not at the end of the directory, return a pointer to
the next entry in the directory. */
while(dir_entry = get_nde(dir_pb))
         /* Check if user = 0xE5 and name, type match */
if (comp_fname(scb,dir_entry) == NAME_EQ)
                   /* Test if this extent has already been
                      unerased */
         if (bv_test(extents,dir_entry -> de_extent))
                   { /* Yes it has */
printf("\n\t\tExtent #%d of %s ignored.",
                            dir_entry -> de_extent,argv[1]);
e; /* Do not unerase this one */
                   continue;
                   3
```

Figure 11-4. (Continued)

3

```
/* Indicate this extent unerased */
         else
                 bv_set(extents,dir_entry -> de_extent);
                 dir_entry -> de_userno = user; /* Unerase entry */
dir_pb.dp_write = 1; /* Need to write sector back */
                 printf("\n\tExtent #%d of %s unerased.",
                         dir_entry -> de_extent,argv[1]);
                 1
        3
3
printf("\n\nFile %s unerased in User Number %d.",
argv[1].user);
bdos(SETDISK,cur_disk); /* Reset to current disk */
build_bv(bv,scb)
                          /* Build bit vector (from directory) */
/* This function scans the directory of the disk specified in
   the directory parameter block (declared as a global variable).
   and builds the specified bit vector, showing all the allocation
blocks used by files matching the name in the search control
   block. */
/* Entry parameters */
struct _bv *bv; /* Pointer to the bit vector */
struct _scb *scb; /* Pointer to search control block */
/* Also uses : directory parameter block (dir_pb) */
struct _bv *bv;
struct _scb *scb;
/* Exit parameters
   The specified bit vector will be created, and will have 1-bits
   set wherever an allocation block is found in a directory
   entry that matches the search control block.
   It also returns the number of directory entries matched. */
£
                          /* Allocation block number */
unsigned abno;
struct _dpb *dpb;
                          /* Pointer to the disk parameter block in the BIOS */
int mcount:
                          /* Match count of dir. entries matched */
mcount = 0:
                          /* Initialize match count */
dpb = get_dpb(dir_pb.dp_disk); /* Get disk parameter block address */
/* make the bit vector with one byte for each eight allocation
   blocks + 1 */
if (!(bv make(bv,(dpb -> dpb_maxabn >>3)+1)))
        printf("\nError -- Insufficient memory to make a bit vector.");
         exit();
         2
/* Set directory to "closed" to force the get_nde
   function to open it. */
dir_pb.dp_open = 0;
/* Now scan the directory building the bit vector */
while(dir_entry = get_nde(dir_pb))
        ī
                                            /* Update match count */
                  ++mcount:
                 for (count = 0;
                                            /* Start with the first alloc. block */
                      count < dir_pb.dp_nabpde; /* For number of alloc. blks. per dir. entry */
                      count++)
                                   /* Set the appropriate bit number for
                                        each nonzero allocation block number */
                          if (dir_pb.dp_nabpde == 8)
                                                            /* assume 8 2-byte numbers */
                                   abno = dir_entry -> _dirab.de_long[count];
                                   1
                                   /* Assume 16 1-byte numbers */
                          else
                                   ŧ
```

Figure 11-4. (Continued)

```
abno = dir_entry -> _dirab.de_short[count];
                           if (abno) bv_set(bv,abno); /* Set the bit */
                  3
         3
return moount;
                           /* Return number of dir. entries matched */
                           /* Check usage */
chk_use(argc)
/* This function checks that the correct number of
   parameters has been specified, outputting instructions
   if not. */
/* Entry parameter */
            /* Count of the number of arguments on the command line */
int argc;
/* The minimum value of argc is 1 (for the program name itself),
   so argc is always one greater than the number of parameters
on the command line */
if (argc == 1 || argc > 3)
         £
         printf("\nUsage :");
         printf("\n\tUNERASE {d:}filename.typ {user}");
         printf("\n\tOnly a single unambiguous file name can be used.)");
exit();
         3
} /* end chk_use */
ssetscb(scb,fname,user,length)  /* Special version of set search control block */
/* This function sets up a search control block according
   to the file name, type, user number, and number of bytes
   to compare.
   The file name can take the following forms :
         filename
         filename.typ
         d:filename.typ
   It sets the bit map according to which disks should be searched.
   For each selected disk, it checks to see if an error is generated when selecting the disk (i.e. if there are disk tables in the BIOS for the disk). */
/* Entry parameters */
                          /* Pointer to search control block */
/* Pointer to the file name */
/* User number to be matched */
struct _scb *scb;
char *fname;
short user;
int length:
                          /* Number of bytes to compare */
/* Exit parameters
   Disk number to be searched. (A = 1, B = 2...)
×/
short disk_id;
                          /* Disk number to search */
setfcb(scb,fname);
                          /* Set search control block as though it
                                were a file control block. */
disk_id = scb -> scb_userno;
                                 /* Set disk_id before it gets overwritten
                                   by the user number */
/* Set user number */
scb -> scb_userno = user;
scb -> scb_length = length;
                                   /* Set number of bytes to compare */
return disk_id;
} /* end setscb */
```

Figure 11-4. (Continued)

A further complication occurs if two or more directory entries of the erased file have the same extent number. This can happen if the file has been created and erased several times. Under these circumstances, UNERASE revives the first entry with a given extent number that it encounters, and displays a message on the console both when an extent is revived and when one is ignored.

Because of the complicated nature of the UNERASE process, the utility can process only a single, unambiguous file name.

The following console dialog shows UNERASE in operation:

P3A>dir *.com<CR> A: UNERASE COM : TEMP2 COM : TEMP3 COM : ERASE COM P3A>unerase<CR> UNERASE Version 1.0 02/12/83 (Library 1.0) Usage : UNERASE {d:}filename.typ {user} Only a single unambiguous file name can be used. P3A>unerase temp1.com<CR> UNERASE Version 1.0 02/12/83 (Library 1.0) Searching disk A. Extent #0 of TEMP1.COM unerased. Extent #0 of TEMP1.COM ignored. File TEMP1.COM unerased in User Number 3. P3A>dir *.com<CR> A: UNERASE COM : TEMP1 COM : TEMP2 COM : TEMP3 COM A: ERASE COM P3A>unerase temp5.com<CR> UNERASE Version 1.0 02/12/83 (Library 1.0) Searching disk A. No directory entries found for file TEMP5.COM.

FIND — Find "Lost" Files

The FIND utility shown in Figure 11-5 searches all user numbers on specified logical disks, matching each entry against an ambiguous file name. It can then display either a disk map showing how many matching files were found in each user number for each disk, or the user number, file name, and type for each matched directory entry.

You can use FIND to locate a specific file or group of files, as shown in the following console dialog:

```
P3B><u>find<CR></u>

FIND Version 1.0 02/11/83 (Library 1.0)

Usage :

FIND d:filename.typ {NAMES}

*:filename.typ (All disks)

ABCD..0P:filename.typ (Selected Disks)

NAMES option shows actual names rather than map.

P3B><u>find ab:*.*<CR></u>

FIND Version 1.0 02/11/83 (Library 1.0)
```

Searching disk : A Searching disk : B Numbers show files in each User Number. --- User Numbers ---Dir. Entries 0 1 з 5 11 12 13 14 15 **Used Free** 2 . . . A: 8 1 1 233 23 772 B: 66 20 74 55 з 252 P3B>find *:*.com<CR> FIND Version 1.0 02/11/83 (Library 1.0) Searching disk : A Searching disk : B Searching disk : C - User Numbers Dir. Entries Ô 1 2 з 5 11 12 13 14 15 Used Free . . . A: 5 23 233 5 Bı 61 4 13 252 772 C -- None --16 112 P3B>find <u>*.com</u> names<CR> FIND Version 1.0 02/11/83 (Library 1.0) Searching disk : B .COM .COM 0/B:CC .COM 0/B:CC2 .COM 0/B:CLINK 2/B:CLIB 1/B:CPM61 . COM 1/B: MOVCPM .COM 1/B:PSWX .COM .COM 0/B:SUBMIT 2/B:CDB . COM 1/B:CPM60 .COM O/B:DDT .COM O/B:EREMOTE .COM 0/B: SPEEDSP . COM 0/B:PIP .COM 0/B: PROTOSP .COM 0/B:RX .COM O/B: TXA .COM O/B: EPUB .COM 0/B:EPRIV O/B:WSC .COM .COM 0/B:X . COM 0/B:CRCK .COM O/B:XSUB .COM O/B:DU .COM 0/B: QERA . COM 0/B: MOVEF .COM .COM 0/B:FINDALL . COM **0/B:REMOTE** 0/B:LOCAL .COM 0/B: DUMP .COM 0/B:MRESET .COM 0/B:ELOCAL .COM 0/B: PUTCPMF5.COM 0/B:FDUMP 0/B: TEST .COM .COM 0/B: INVIS .COM . COM .COM 0/B:L80 0/B:LIST 0/B:PUB .COM O/B:LOAD .COM O/B:MAC .COM 0/B: SCRUB . COM 0/B:RXA .COM 0/B:STAT . COM 0/B: TX .COM 0/B: ERASEALL. COM 0/B:WM .COM O/B:MSFORMAT.COM 0/B: UNERA 0/B:STATUS .COM .COM .COM 0/B:MSINIT .COM O/B:VIS .COM O/B: DDUMP .COM 0/B:WSVTIP . COM O/B:NEWVE .COM 0/B: XD .COM 0/B:PRIV . COM 0/B:FORMATMA.COM . COM 0/B:FCOMP O/B: DDUMPA 0/B: PUTSYS1C. COM 0/B:DDUMPNI . COM O/B:DSTAT .COM O/B:ASM .COM 2/B:CDBTEST .COM 0/B:OLDSYS .COM 0/B:E .COM 2/B:F/C .COM .COM 3/B:DATE 3/B:ERASE 3/B: FUNKEY . COM .COM 3/B:FIND .COM Press Space Bar to continue.... 3/B: SPACE . COM 3/B:UNERASE .COM 3/B: MAKE .COM 3/B: MOVE . COM 1/B: PUTSYSWX, COM 3/B:TIME .COM 3/B:ASSIGN . COM 3/B: SPEED .COM 3/B: PROTOCOL, COM **0/B:PRINTC** .COM 3/B:T .COM

#define VN "1.0 02/11/83"	
/* FIND - This utility can display either a map showing on which disks and in which user numbers files matching the specified ambiguous file name are found, or the actual names matched. */	
<pre>#include <library.h></library.h></pre>	
struct _dirpb dir_pb;	/* Directory management parameter block */
struct _dir *dir_entry;	<pre>/* Pointer to directory entry (somewhere in dir_pb) */</pre>
struct _scb scb;	/* Search control block */
<pre>char file_name[20];</pre>	/* Formatted for display : un/d:FILENAME.TYP */

Figure 11-5. FIND.C, a utility program that locates specific files or groups of files

```
short cur_disk;
                                     /* Current logical disk at start of program */
                                     /* Match count (no. of file names matched) */
  int mcount;
                                     /* Per disk match count */
  int dmcount;
  int lcount;
                                      /* Line count (for lines displayed) */
  int map_flag;
                                     /* 0 = show file names of matched files,
                                           NZ = show map of number of files */
           /* The array below is used to tabulate the results for each
disk drive, and for each user number on the drive.
In addition, two extra "users" have been added for "free"
              and "used" values. */
                                    /* Disk A -> P, ysers O -> 13, free, used */
/* "User" number for used entities */
/* "User" number for free entities */
  unsigned disk_map[16][18];
  #define USED_COUNT 16
#define FREE COUNT 17
  main(argc,argv)
                           /* Argument count */
/* Argument vector (pointer to an array of chars.) */
  short arget
  char #argv[]:
  cur_disk = bdos(GETDISK);
                                     /* Get current default disk */
  dm_clr(disk_map);
                                     /* Reset disk map */
           /* Set search control block
              disks, name, type, user number, extent number,
and number of bytes to compare -- in this case, match all users,
  but only extent 0 */
setscb(scb.argv[1].'?'.0.13); /* Set disks, name, type */
  map_flag = usstrcmp("NAMES", argv[2]); /* Set flag for map option */
                                              /* Initialize counts */
  lcount = dmcount = mcount = 0;
       (scb.scb_disk = 0; /* Starting with logical disk A: */
scb.scb_disk < 16; /* Until logical disk P: */
scb.scb_disk++) /* Move to next logical disk */
  for (scb.scb_disk = 0;
  ş
           /* Check if current disk has been selected for search */
  if (!(scb.scb_adisks & (1 << scb.scb_disk)))
continue; /* No,so bypass this disk */
  - lcount++:
  if (!map_flag)
  /* Set the directory to "closed", and force the get_nde
     function to open it */
  dir_pb.dp_open = 0;
           /* While not at the end of the directory, set a pointer to the
              next directory entry */
  while(dir_entry = get_nde(dir_pb))
           /* Check if entry in use, to update
              the free/used counts */
           if (dir_entry -> de_userno == 0xE5)
                                                        /* Unused */
                    disk_map[scb.scb_disk][FREE_COUNT]++;
           e15e
                    /* In use */
                    disk_map[scb.scb_disk][USED_COUNT]++;
           /* Select only those active entries that are the
              first extent (numbered 0) of a file that matches the name supplied by the user */
```

Figure 11-5. (Continued)

```
if (
               .
(dir_entry -> de_userno != 0xE5) &&
(dir_entry -> de_extent == 0) &&
(comp_fname(scb,dir_entry) == NAME_EQ)
              )
                    £
                    mcount++:
                                       /* Update matched counts */
                    dmcount++;
                                       /* Per disk count */
                    if (map_flag)
                                       /* Check map option */
                              ş
                                        /* Update disk map */
                              disk_map[scb.scb_disk][dir_entry -> de_userno]++;
                                        /* Display names */
                    else
                              conv_dfname(scb.scb_disk,dir_entry,file_name);
printf("%s ",file_name);
                              /* Check if need to start new line */
if (!(dmcount % 4))
                                        ş
                                        putchar(^\n');
                                                 if (++1count > 18)
                                                           lcount = 0;
printf("\nPress Space Bar to continue....");
                                                           getchar();
                                                           putchar(^\n');
                                                 3
                                       3
                              1
                    } /* End of directory */
          } /* All disks searched */
if (map_flag)
printf("\n
                              Numbers show files in each user number.");
--- User Numbers ---
printf("\n
                                                                                               Dir. Entries");
                                                 /¥ Display disk map ★/
dm_disp(disk_map,scb.scb_adisks);
if (mcount == 0)
printf("\n --- File Not Found --- ");
bdos(SETDISK,cur_disk); /* Reset to current disk */
chk use(argc)
                             /* check usage */
/* This function checks that the correct number of
parameters has been specified, outputting instructions
   if not.
×/
/* Entry parameter */
int argc; /* Count of the number of arguments on the command line */
/* The minimum value of argc is 1 (for the program name itself),
   so argc is always one greater than the number of parameters
   on the command line */
if (argc == 1 !! argc > 3)
printf("\nUsage :");
printf("\n\tFIND d:filename.typ {NAMES}");
printf("\n\t *:filename.typ (All disks)");
printf("\n\t ABCD..OP:filename.typ (Selected Disks)");
printf("\n\tNAMES option shows actual names rather than map.");
exit();
3
```

Figure 11-5. (Continued)

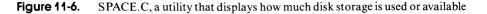
SPACE – Show Used Disk Space

The SPACE utility shown in Figure 11-6 scans the specified logical disks and displays a disk map that shows, for each user number on each logical disk, how many Kbytes of storage have been used. It also displays the total number of Kbytes used and free on each logical disk.

Here is an example console dialog showing SPACE in operation:

```
P3B>space<CR>
SPACE Version 1.0 02/11/83 (Library 1.0)
Usage :
        SPACE *
                       (All disks)
       SPACE ABCD.. OP (Selected Disks)
                           1
P3B>space *<CR>
SPACE Version 1.0 02/11/83 (Library 1.0)
Searching disk : A
Searching disk : B
Searching disk : C
                 Numbers show space used in kilobytes.
                         --- User Numbers ---
                                                          Space (Kb)
                        5 ... 10 11 12 13 14 15 Used Free
    0
        1
            2
                з
                     4
A: 18 202
                38
                                                           258 1196
B: 692 432 656 548 36
                                                          2364
                                                                996
                                                           140 204
C: 140
```

```
#define VN "1.0 02/11/83"
/* SPACE -- This utility displays a map showing on the amount of space
(expressed as relative percentages) occupied in each user number
    for each logical disk. It also shows the relative amount of space
    free. */
#include <LIBRARY.H>
                                     /* Directory management parameter block */
struct _dirpb dir_pb;
struct _dir *dir_entry;
struct _scb scb;
struct _dpb dpb;
                                       /* Pointer to directory entry */
/* Search control block */
                                       /* CP/M's disk parameter block */
                                        /* Formatted for display : un/d:FILENAME.TYP */
char file_name[20];
short cur_disk;
                           /* Current logical disk at start of program
                                 NZ = show map of number of files */
                              /* Used to access the allocation block numbers
int count;
                                 in each directory entry */
                              /* Used to access the disk map when calculating */
int user:
/* The array below is used to tabulate the results for each
   disk drive, and for each user number on the drive.
In addition, two extra "users" have been added for "free"
   and "used" values.
¥/
                                      /* Disk A -> P, users 0 -> 15, free, used */
unsigned disk map[16][18]:
                                       /* Disk H =/ 1, users 0 / 10, 112,
/* "User" number for used entities */
/* "User" number for free entities */
#define USED_COUNT 16
#define FREE_COUNT 17
main(argc,argv)
short arge;
                             /* Argument count */
                             /* Argument vector (pointer to an array of chars.) */
char #argv[];
```



```
printf("\nSPACE Version %s (Library %s)", VN, LIBVN);
                                    /* Check usage */
 chk use(argc);
cur_disk = bdos(GETDISK);
                                     /* Get current default disk */
 dm_clr(disk_map);
                                     /* Reset disk map */
                                     /* Special version : set disks,
ssetscb(scb,argv[1]);
                                        name, type */
for (scb.scb_disk = 0;
                                     /* Starting with logical disk A: */
      scb.scb_disk < 16;</pre>
                                     /* Until logical disk P: */
      scb.scb_disk++)
                                     /* Move to next logical disk */
          £
          /* Check if current disk has been selected for search */
          if (!(scb.scb_adisks & (1 << scb.scb_disk)))
                                    /* No, so bypass this disk */
                   continue:
         printf("\nSearching disk : %c",(scb.scb_disk + ^A'));
dir_pb.dp_disk = scb.scb_disk; /* Set to disk to be searched */
          /* Set the directory to "closed", and force the get_nde
         function to open it */
dir_pb.dp_open = 0;
          /* While not at the end of the directory, set a pointer
             to the next entry in the directory */
         while (dir_entry = get_nde(dir_pb))
                   if (dir_entry -> de_userno == 0xE5)
                            continue:
                                             /* Bypass inactive entries */
                                              /* Start with the first alloc. block */
                   for (count = 0;
                        count < dir_pb.dp_nabpde; /* For number of alloc. blks. per dir. entry */
                        count++)
                            if (dir_pb.dp_nabpde == 8)
                                                                /* Assume 8 2-byte numbers */
                                     disk_map[scb.scb_disk][dir_entry ->_de_userno]
                                              += (dir_entry -> _dirab.de_long[count] > 0 ? 1 : 0);
                            else
                                     /* Assume 16 1-byte numbers */
                                     f
                                     disk_map[scb.scb_disk][dir_entry -> de_userno]
+= (dir_entry -> _dirab.de_short[count] > 0 ? 1 : 0);
                                     3
                                     /* All allocation blocks processed */
                  3
                            /* End of directory for this disk */
         /* Compute the storage used by multiplying the number of
             allocation blocks counted by the number of Kbytes in each allocation block. */
         for (user = 0; /* Start with user 0 */
               user < 16; /* End with user 15 */
               user ++)
                           /* Move to next user number */
                            /* Compute size occupied in Kbytes */
                  disk_map[scb.scb_disk][user] *= dir_pb.dp_absize;
                           /* Build up sum for this disk */
                  disk_map[scb.scb_disk][USED_COUNT] += disk_map[scb.scb_disk][user];
         /* Free space = (# of alloc. blks * # of kbyte per blk)
                  – used Kbytes
         - used Nories
- (directory entries * 32) / 1024 ... or divide by 32 */
disk_map[scb.scb_disk]IFREE_COUNT] = (dir_pb.dp_nab * dir_pb.dp_absize)
- disk_map[scb.scb_disk]IUSED_COUNT]
- (dir_pb.dp_nument >> 5); /* Same as / 32 */
         3
                  /* All disks processed */
printf("\n
                                Numbers show space used in kilobytes.");
printf("\n
                                           -- User Numbers
                                                                                           Space (Kb)");
dm_disp(disk_map,scb.scb_adisks);
                                             /* Display disk map */
```

Figure 11-6. (Continued)

```
bdos(SETDISK,cur_disk); /* Reset to current disk */
ssetscb(scb,ldisks)
                           /* Special version of set search control block */
/* This function sets up a search control block according
    to just the logical disks specified. The disk are specified as
   a single string of characters without any separators. An asterisk means "all disks." For example --
         ABGH
                   (disks A:, B:, G: and H: )
(all disks for which SELDSK has tables)
   It sets the bit map according to which disks should be searched.
   For each selected disk, it checks to see if an error is generated
when selecting the disk (i.e. if there are disk tables in the BIOS
   for the disk).
   The file name, type, and extent number are all set to "?" to match all possible entries in the directory. */
/* Entry parameters */
                           /* Pointer to search control block */
/* Pointer to the logical disks */
struct _scb *scb;
char *ldisks;
                                                                                 ١
/* Exit parameters
   None.
×/
                            /* Disk number currently being checked */
int disk:
                            /* Bit map for active disks */
unsigned adisks;
adisks = 0;
                            /* Assume no disks to search */
if (*ldisks)
                            /* Some values specified */
         if (*ldisks == '*')
                                    /* Check if "all disks" */
                   adisks = 0xFFFF;
                                              /* Set all bits */
                   3
                                      /* Set specific disks */
         else
                   Ŧ
                   while(*ldisks) /* Until end of disks reached */
                            /* Build the bit map by getting the next disk
                                id. (A - P), converting it to a number
in the range 0 - 15, and shifting a 1-bit
                                left that many places and OR ing it into
                                the current active disks.
                            ×/
                            adisks != 1 << (toupper(*ldisks) - 'A');</pre>
                            ++ldisks;
                                               /* Move to next character */
                            ł
                   3
         ł
else
         /* Use only current default disk */
         /* Set just the bit corresponding to the current disk */
         adisks = 1 << bdos(GETDISK);</pre>
         /* Set the user number, file name, type, and extent to "?"
            so that all active directory entries will match */
/* 0123456789012 */
strcpy(&scb -> scb_userno,"???????????);
          /* Make calls to the BIOS SELDSK routine to make sure that
             all of the active disk drives have disk tables for them
in the BIOS. If they don't, turn off the corresponding
             bits in the bit map. */
for (disk = 0;
                           /* Start with disk A: */
                            /* Until disk P: */
      disk < 16;
                            /* Use next disk */
      disk++)
          if ( !((1 << disk) & adisks))
                                               /* Avoid selecting unspecified disks */
                   continue;
```

Figure 11-6. (Continued)

```
if (biosh(SELDSK,disk) == 0)
                                             /* Make BIOS SELDSK call */
                                              /* Returns O if invalid disk */
                  /* Turn OFF corresponding bit in mask
                     by AND-ing it with bit mask having
                  all the other bits set = 1. */
adisks &= ((1 << disk) * 0xFFFF);
         3
scb -> scb_adisks = adisks;
                                    /* Set bit map in scb */
} /* End ssetscb */
                            /* Check usage */
chk use(argc)
/* This function checks that the correct number of
   parameters has been specified, outputting instructions
   if not. */
/* Entry parameter */
                 /* Count of the number of arguments on the command line */
int arge:
£
         /* The minimum value of argc is 1 (for the program name itself),
            so argc is always one greater than the number of parameters
on the command line */
if (argc != 2)
         £
         printf("\nUsage :");
printf("\n\tSPACE * (All disks)");
printf("\n\tSPACE ABCD..OP (Selected Disks)");
         exit();
         3
} /* End chk_use */
```

Figure 11-6. (Continued)

MOVE – Move Files Between User Numbers

The MOVE utility shown in Figure 11-7 moves files from one user number to another on the same logical disk. The movement is achieved by changing the user number in all the relevant directory entries. This is much faster than copying the files. It also avoids having multiple copies of the same file on the disk.

Here is a console dialog showing MOVE in operation:

```
P3B>move<CR>
MOVE Version 1.0 02/10/83 (Library 1.0)
Usage :
        MOVE d:filename.typ to_user {from_user} {NAMES}
             *:filename.typ (All disks)
             ABCD..OP:filename.typ (Selected Disks)
        NAMES option shows names of files moved.
P3B>dir *.com<CR>
B: ERASE
           COM : FUNKEY COM : DATE
                                       COM : FIND
                                                      COM
           COM : UNERASE COM : MAKE
B: SPACE
                                       COM : MOVE
                                                      COM
B: TIME
          COM : ASSIGN COM : SPEED
                                       COM : PROTOCOL COM
P3B>move *.com 0 names<CR>
MOVE Version 1.0 02/10/83 (Library 1.0)
Moving file(s) 3/B:?????.COM -> User 0.
```

. COM 0/B:FUNKEY .COM 0/B:DATE 0/B:UNERASE .COM 0/B:MAKE .COM 0/B:FIND .COM 0/B:ERASE . COM .COM .COM 0/B:SPACE 0/B: MOVE 0/B:ASSIGN .COM 0/B:SPEED .COM 0/B:TIME . COM 0/B:PROTOCOL.COM P3B>user O<CR> POB>dir B: ERASE COM : FUNKEY COM : DATE COM : FIND COM COM : UNERASE COM : MAKE B: SPACE COM : MOVE COM B: TIME COM : ASSIGN COM : SPEED COM : PROTOCOL COM

```
#define VN "1.0 02/10/83"
/* MOVE -- This utility transfers file(s) from one user number to
another, but on the SAME logical disk. Files are not actually
   copied -- rather, their directory entries are changed. */
#include <LIBRARY.H>
struct _dirpb dir_pb;
                                    /* Directory management parameter block */
struct _dir *dir_entry;
struct _scb scb;
                                    /* Pointer to directory entry */
/* Search control block */
#define DIR_BSZ 128
                                    /* Directory buffer size */
char dir_buffer[DIR_BSZ];
                                    /* Directory buffer */
char file_name[20];
                                    /* Formatted for display : un/d:FILENAME.TYP */
                                    /* NZ to display names of files moved */
short name_flag;
short cur_disk;
                                    /* Current logical disk at start of program */
                                    /* User number from which to move files */
/* User number to which files will be moved */
int from user;
int to_user;
int mcount;
                                    /* Match count (no. of file names matched) */
int dmcount;
                                     /* Per-disk match count */
int lcount;
                                    /* Line count (for lines displayed) */
main(argc,argv)
                           /* Argument count */
/* Argument vector (pointer to an array of chars.) */
short arge:
char *argv[];
printf("\nMOVE Version %s (Library %s)", VN, LIBVN);
chk_use(argc);
                                    /* Check usage */
to_user = atoi(argv[2]);
                                    /* Convert user no. to integer */
/* Set and check destination user number */
if(to_user > 15)
         £
         printf("\nError -- the destination user number cannot be greater than 15.");
/* Set the current user number */
from_user = bdos(GETUSER,0xFF);
         /* Check if source user number specified */
if (isdigit(argv[3][0]))
         £
                  /* Set and check source user number */
         if((from_user = atoi(argv[3])) > 15)
                  £
                  printf("\nError -- the source user number cannot be greater than 15.");
                  exit();
                  3
                  /* Set name suppress flag from parameter #4 */
         name_flag = usstrcmp("NAMES",argv[4]);
         - 1
else
                  /* No source user specified */
         ŧ
```

Figure 11-7. MOVE.C, a utility program that changes files' user numbers

```
/* Set name suppress flag from parameter #3 */
         name_flag = usstrcmp("NAMES", argv[3]);
/* To simplify the logic below, name_flag must be made
NZ if it is equal to NAME_EQ, 0 if it is any other value */
name_flag = (name_flag == NAME_EQ ? 1 : 0);
if (to_user == from_user)
                                     /* To = from */
         printf("\nError - 'to' user number is the same as 'from' user number.");
         exit();
         /* Set the search control block file name, type, user number,
extent number, and length -- length matches user number, file
             name, and type. As the extent number does not enter into the comparison, all extents of a given file will be found. */
setscb(scb, argv[1], from_user, '?', 13);
cur_disk = bdos(GETDISK);  /* Get current default disk */
lcount = dmcount = mcount = 0;  /* Initialize counts */
for (scb.scb_disk = 0;
                                     /* Starting with logical disk A: */
                                      /* Until logical disk P: */
      scb.scb_disk < 16;</pre>
                                      /* Move to next logical disk */
      scb.scb_disk++)
         £
                   /* Check if current disk has been selected for search */
         if (!(scb.scb_adisks & (1 << scb.scb_disk)))
continue; /* No, so bypass this disk */
                   /* convert search user number and name for output */
         conv_dfname(scb.scb_disk,scb,file_name);
printf("\n\nMoving file(s) %s -> User %d.",file_name,to_user);
                                      /* Update line count */
         lcount++:
         dir_pb.dp_disk = scb.scb_disk; /* Set to disk to be searched*/
                                                /* Reset disk matched count */
         dmcount = 0:
         if (name_flag)
                                      /* If file names are to be displayed */
                   putchar('\n'); /* Move to column 1 */
                   /* Set the directory to "closed" to force the get_nde
                      function to open it. */
         dir_pb.dp_open = 0;
                   /* While not at the end of the directory, set a pointer
to the next directory entry */
         while(dir_entry = get_nde(dir_pb))
                   £
                             /* Match those entries that have the correct
                                user number, file name, type, and any extent number. */
                   if (
                        (dir_entry -> de_userno != 0xE5) &&
                        (comp_fname(scb,dir_entry) == NAME_EQ)
                       )
                             £
                             dir_entry -> de_userno = to_user;
                                                                            /* Move to new user */
                                       /* Request sector to be written back */
                             dir_pb.dp_write = 1;
                                               /* Update matched counts */
/* Per-disk count */
                             mcount++:
                             dmcount++:
                             if (name_flag) /* Check map option */
                                      /* Check if need to start new line */
                                      if (!(dmcount % 4))
                                                ÷
                                                putchar((\n');
                                                if (++1count > 18)
```

Figure 11-7. (Continued)

```
lcount = 0;
                                                         printf("\nPress Space Bar to continue....");
                                                         getchar();
                                                         putchar (^{n});
                                                3
                                      1
                            1
                   3
         3
if (meant == 0)
         printf("\n --- No Files Moved --- ");
bdos(SETDISK,cur_disk); /* Reset to current disk */
chk_use(argc)
                             /* Check usage */
/* This function checks that the correct number of
   parameters has been specified, outputting instructions
   if not */
/* Entry parameter */
                  /* Count of the number of arguments on the command line */
int argc;
ş
/* The minimum value of argc is 1 (for<sub>1</sub> the program name itself),
so argc is always one greater than the number of parameters
on the command line */
if (argc == 1 || argc > 5)
         printf("\nUsage :");
         printf("\n\tMOVE d:filename.typ to_user {from_user} {NAMES}");
         printf("\n\t *:filename.typ (All disks)");
printf("\n\t ABCD..OP:filename.typ (Selected Disks)");
         printf("\n\tNAMES option shows names of files moved.");
         exit();
3
```

Figure 11-7. (Continued)

Other Utilities

The utility programs described in this section are by no means a complete set. You may want to develop many other specialized utility programs. Some possibilities are:

FILECOPY

A more specialized version of PIP could copy ambiguously specified groups of files. Of special importance would be the ability to read a file containing the names of the files to be copied. A useful option would be the ability to detect the setting of the unused file attribute bit and copy only files that have been changed.

PROTECT/UNPROTECT

This pair of utilities would allow you to "hide" files in user numbers greater than 15. Files so hidden could not be accessed other than by UNPRO-TECTing them, thereby moving them back into the normal user number range.

RECLAIM

This utility would read all sectors on a disk (using the BIOS). Any bad sectors encountered could then be logically removed by creating an entry in the file directory, with allocation block numbers that would effectively "reserve" the blocks containing the bad sectors.

OWNER

This utility, given a track or sector number, would access the directory and determine which file or files were using that part of the disk. This is useful if you have a bad sector or track on a disk. You then can determine which files have been damaged.

Utility Programs for the Enhanced BIOS

This section describes several utility programs that work with the enhanced BIOS shown in Figure 8-10. Several of these utilities work directly with the physical devices on the computer system, which can vary from computer to computer. The library header contains #define declarations for device numbers and names for physical devices (Figure 11-2, f and Figure 11-2, g).

These #define statements are used to build a physical-device code table. If you have more physical devices or want to change the names by which you refer to the devices, you will need to change these definitions.

All of these utilities share some common features in the way that they are invoked. If they are called without any parameters, they display instructions on the console regarding what parameters are available. If they are called with the word "SHOW" (or "S", "SH", and so forth) as a parameter, they display the current settings of whatever attribute the utility controls.

MAKE -- Make Files "Invisible" or "Visible"

The MAKE utility shown in Figure 11-8 is designed to operate in conjunction with the public files option implemented in the enhanced BIOS of Figure 8-10. It has two modes of operation—making files "invisible" or "visible."

An invisible file is one in user 0 which has been set to Read-Only and System status. When the public files option is enabled, these files cannot be seen when you use the DIR command, nor can they be erased accidentally.

A visible file is one that has been set to Read/Write and Directory status.

When files are made invisible, they are transferred from the current user number to user 0. When files are made visible, they are transferred from user 0 to the current user number.

Here is an example console dialog showing MAKE in operation:

```
P3B><u>make<CR></u>
MAKE Version 1.0 02/12/83 (Library 1.0)
```

Usage : MAKE d:filename.typ INVISIBLE {NAMES} VISIBLE *:filename.typ (All disks) ABCD..OP:filename.typ (Selected Disks) NAMES option shows names of files processed. P3B>dir *.com<CR> COM : UNERASE COM : ASSIGN COM : PROTOCOL COM B: ERASE P3B>make *.com invisible names<CR> MAKE Version 1.0 02/12/83 (Library 1.0) Moving files from User 3 to 0 and making them Invisible. Searching disk : B .COM made Invisible in User 0. 0/B:ERASE O/B:UNERASE .COM made Invisible in User 0. O/B:ASSIGN .COM made Invisible in User 0. O/B:PROTOCOL.COM made Invisible in User 0. P3B>make erase.com visible names<CR> MAKE Version 1.0 02/12/83 (Library 1.0) Moving files from User 0 to 3 and making them Visible. Searching disk : B .COM made Visible in User 3. 3/B: FRASE #define VN "1.0 02/12/83" /* MAKE - This utility is really two very similar programs; which one depends on the parameter specified on the command line. INVISIBLE finds all of the specified files, moves them to user number 0, and sets them to be System and Read Only status. These files can then be accessed from user numbers other than 0 when the public files feature is enabled in the BIOS. VISIBLE is the opposite in that the specified files are and Read/Write status. */ #include <LIBRARY.H> struct _dirpb dir_pb; /* Directory management parameter block */ /* Pointer to directory entry */ struct _dir *dir_entry; struct _scb scb; short to_user; short from_user; /* Search control block */ /* User number to which files will be set */
/* User number from which files will be moved */ char file_name[20]; /* Formatted for display : un/d:FILENAME.TYP */ short name flag; /* NZ to display names of files moved */ short cur_disk; /* Current logical disk at start of program */ int mcount; /* Match count (no. of file names matched) */ /* NZ when parameter specifies invisible */ /* Pointer to either "invisible" or "visible" */ short invisible; char *operation: main(argc.argv) /* Argument count */ /* Argument vector (pointer to an array of chars.) */ short argc; char *argv[];

Figure 11-8. MAKE.C, a utility that makes files "invisible" and protected or makes them "visible," accessible, and unprotected

```
cur_disk = bdos(GETDISK);
mcount = 0;
                                    /* Initialize count */
         /* Set the invisible flag according to the parameter */
invisible = usstrcmp("VISIBLE", argv[2]);
         /* Set the from_user and to_user numbers depending on which
            program is to be built, and the parameters specified. */
if (invisible)
         from_user = bdos(GETUSER,0xFF); /* Get current user number */
                        /* Always move files to user 0 */
         to user = 0;
         operation = "Invisible";
                                          /* Set pointer to string */
         /* visible */
else
         from_user = 0;
                                            /* Always move from user 0 */
         to_user = bdos(GETUSER, 0xFF);
                                           /* Get current user */
         operation = "Visible";
                                            /* Set pointer to string */
         /* Set search control block disks, name, type, user number,
extent number, and number of bytes to compare -- in this
case, match the "from" user, all extents. */
setscb(scb,argv[1],from_user,??,13); /* Set disks, name, type */
name_flag = usstrcmp("NAMES",argv[3]); /* Set name-suppress flag from param. 3 */
         /* To simplify the logic below, name_flag must be made
NZ if it is equal to NAME_EQ, 0 if it is any other value */
name_flag = (name_flag == NAME_EQ ? 1 : 0);
         /* Convert search user number and name for output */
conv_dfname(scb.scb_disk,scb,file_name);
printf("\n\nMoving files from User %d to %d and making them %s.",
        from_user, to_user, operation);
for (scb.scb_disk = 0;
                                  /* Starting with logical disk A: */
     scb.scb_disk < 16;</pre>
                                 /* Until logical disk P: */
     scb.scb_disk++)
                                  /* Move to next logical disk */
                 /* Check if current disk has been selected for search */
         printf("\nSearching disk : %c",(scb.scb_disk + 'A'));
        dir_pb.dp_disk = scb.scb_disk; /* Set to disk to be searched*/
                                   /* If file names are to be displayed */
         if (name flag)
                 putchar('\n'); /* Move to column 1 */
                  /* Set the directory to "closed", and force the get_nde
                    function to open it. */
        dir_pb.dp_open = 0;
                 /* While not at the end of the directory,
set a pointer to the next directory entry. */
        while(dir_entry = get_nde(dir_pb))
                 £
                          /* Match those entries that have the correct
user number, file name, type, and any
extent number. */
                 if (
                     (dir_entry -> de_userno != 0xE5) &&
(comp_fname(scb,dir_entry) == NAME_EQ)
                     ì
                          ş
```

Figure 11-8. (Continued)

Ŧ

```
mcount++;
                                                /* Update matched counts */
                             if (invisible)
                                       { /* Set ms bits */
dir_entry -> de_fname[8] != 0x80;
dir_entry -> de_fname[9] != 0x80;
                             else
                                       /* Visible */
                                       { /* Clear ms bits */
dir_entry -> de_fname[8] &= 0x7F;
dir_entry -> de_fname[9] &= 0x7F;
                                       /* Move to correct user number */
                             dir_entry -> de_userno = to_user;
                                       /* Indicate sector to be written back */
                             dir_pb.dp_write ≈ 1;
                                       /* Check if name to be displayed */
                             if (name_flag)
                                       Ŧ
                                       conv_dfname(scb.scb_disk,dir_entry,file_name);
                                       printf("\n\t%s made %s in User %d.",
                                                file_name,operation,to_user);
                                       э.
                             3
                   } /* All directory entries processed */
/* All disks processed */
         ł
if (mcount == 0)
         printf("\n --- No Files Processed --- ");
bdos(SETDISK,cur_disk); /* Reset to current disk */
chk_use(argc)
                            /* Check usage */
/* This function checks that the correct number of
   parameters has been specified, outputting instructions
   if not.
¥/
/* Entry parameter */
/* Entry parameter */
int argc; /* Count of the number of arguments on the command line */
£
         /* The minimum value of argc is 1 (for the program name itself),
             so argc is always one greater than the number of parameters on the command line \ast\prime
if (argc == 3 || argc == 4)
         returns
else
          $
         printf("\nUsage :");
         printf("\n\tMAKE d:filename.typ INVISIBLE {NAMES}");
         printf("\n\t
                                                VISIBLE");
         printf("\n\t
                              *:filename.typ (All disks)");
         printf("\n\t ABCD..OP;filename.typ (Selected Disks)");
printf("\n\tNAMES option shows names of files processed.");
         exit();
3
```

Figure 11-8. (Continued)

SPEED — Set Baud Rates

The SPEED utility shown in Figure 11-9 sets the baud rate for a specific serial device. Here is an example console dialog that shows several of the options:

P3B>speed<CR> SPEED 1.0 02/17/83 The SPEED utility sets the baud rate speed for each physical device. SPEED physical-device baud-rate, or Usage is : SPEED SHOW (to show current settings) Valid physical devices are: TERMINAL PRINTER MODEM Valid baud rates are: 300 600 1200 2400 4800 9600 19200 P3B>speed show<CR> SPEED 1.0 02/17/83 Current Baud Rate settings are : TERMINAL set to 9600 baud. PRINTER set to 9600 baud. MODEM set to 9600 baud. P3B>speed m 19<CR> SPEED 1.0 02/17/83 Current Baud Rate settings are : TERMINAL set to 9600 baud. PRINTER set to 9600 baud. MODEM set to 19200 baud. P3B>speed xyz 12<CR> SPEED 1.0 02/17/83 Physical Device 'XYZ' is invalid or ambiguous. Legal Physical Devices are : TERMINAL PRINTER MODEM #define VN "\nSPEED 1.0 02/17/83" /* This utility sets the baud rate speed for each of the physical devices. */

#include <LIBRARY.H>

struct _ct ct_pdev[MAXPDEV + 2]; /* Physical device table */
 /* Hardware specific items */

1

Figure 11-9. SPEED.C, a utility that sets the baud rate for a specific device

```
/* Baud rates for serial ports */
#define B300
                  0x35
                                     /* 300 baud */
/* 600 baud */
#define B600
                  0x36
                                     /* 1200 baud */
#define B1200
                  0x37
                                     /* 2400 baud */
/* 4800 baud */
#define B2400
                  0x3A
#define B4800
                  0x3C
#define B9600
                                     /* 9600 baud */
                  0x3E
#define B19200 0x3F
                                     /* 19200 baud */
struct _ct ct_br[10]; /* Code table for baud rates (+ spare entries) */
         /* Parameters on the command line */
#define PDEV argv[1] /* Physical device */
#define BAUD argv[2] /* Baud rate */
main(argc,argv)
int argc;
char *argv[];
printf(VN); /* Display sign-on message */
setup(); /* Set up code tables */
chk_use(argc); /* Check correct usage */
         /* Check if request to show current settings */
if (usstrcmp("SHOW",argv[1]))
                           /* No -- assume setting is required */
         £
         set_baud(get_pdev(PDEV),get_baud(BAUD)); /* Set baud rate */
show_baud();
                            /* Display current settings */
} /* end of program */
setup()
                            /* set up the code tables for this program */
ŧ
         /* Initialize the physical device table */
ct_init(ct_pdev[0],T_DEVN,PN_T); /* Terminal */
ct_init(ct_pdev[1],P_DEVN,PN_P); /* Printer */
ct_init(ct_pdev[2],M_DEVN,PN_M); /* Modem */
ct_init(ct_pdev[3],CT_SNF,"*"); /* Terminator */
         /* Initialize the baud rate table */
ct_init(ct_br[0],B300,"300");
ct_init(ct_br[1],B600,"600");
ct_init(ct_br[2],B1200,"1200");
ct_init(ct_br[3],B2400,"2400");
ct_init(ct_br[4],B4800,"4800");
ct_init(ct_br[5],B9600,"9600");
unsigned
get_pdev(ppdev) /* Get physical device */
/* This function returns the physical device code
  specified by the user in the command line. */
char *ppdev;
                           /* Pointer to character string */
unsigned retval;
                                              /* Return value */
/* Get code for ASCII string */
if (retval == CT_SNF)
          £
         printf("\n\007Physical Device '%s' is invalid or ambiguous.",
         ppdev);
printf("\nLegal Physical Devices are : ");
         ct_disps(ct_pdev); /* Display all values */
         exit();
         3
return retval;
                                    /* Return code */
unsigned
get_baud(pbaud)
/* This function returns the baud rate time constant for
    the baud rate specified by the user in the command line */
```

١

Figure 11-9. (Continued)

```
char *pbaud;
                         /* Pointer to character string */
4
unsigned retval;
                                           /* Return value */
retval = ct_parc(ct_br,pbaud); /* Get code for ASCII string */
if (retval == CT_SNF)
                                  /* If string not found */
         printf("\n\007Baud Rate '%s' is invalid or ambiguous.",
         return retval;
                         /* Return code */
set_baud(pdevc,baudc)
                          /* Set the baud rate of the specified device */
                          /* Physical device code */
int pdevc;
short baudc;
                          /* Baud rate code */
                          /* On some systems this may have to be a
                             two-byte (unsigned) value */
short *baud_rc;
                          /* Pointer to the baud rate constant */
                          /* On some systems this may have to be a
                             two-byte (unsigned) value */
(* Note: the respective codes for accessing the baud rate constants
via the get_cba (get configuration block address) function are:
Device #0 = 19, #1 = 21, #2 = 23. This function uses this
   mathematical relationship */
         /* Set up pointer to the baud rate constant */
baud_rc = get_cba(CB_D0_BRC + (pdevc << 1));</pre>
        /* Then set the baud rate constant */
*baud_rc = baudc;
         /* Then call the BIOS initialization routine */
bios(CIOINIT, pdevc);
show_baud()
                         /* Show current baud rate */
int pdevn:
                         /* Physical device number */
short baude:
                          /* Baud rate code */
                          /* On some systems this may have to be a
                          two-byte (unsigned) value */
/* Pointer to the baud rate constant */
/* On some systems this may have to be a
short *baud rc;
                             two-byte (unsigned) value */
mathematical relationship */
printf("\nCurrent baud rate settings are :");
for (pdevn = 0; pdevn <= MAXPDEV; pdevn ++) /* All physical devices */
                 /* Set up pointer to the baud rate constant --
                    the code for the get_cba function is computed
                    by adding the physical device number *2 to
                    the Baud Rate code for device #0 */
        baud_rc = get_cba(CB_D0_BRC + (pdevn << 1));</pre>
                 /* Then set the baud rate constant */
        baude = *baud_re;
        printf("\n\t%s set to %s baud.",
                 ct_strc(ct_pdev,pdevn), /* Get ptr. to device name */
ct_strc(ct_br,baudc) ); /* Get ptr. to baud rate */
        3
ł
chk_use(argc)
                         /* Check correct usage */
int argc;
                         /* Argument count */
£
```

```
Figure 11-9. (Continued)
```

Figure 11-9. (Continued)

PROTOCOL — Set Serial Line Protocols

The PROTOCOL utility shown in Figure 11-10 is used to set the protocol for a specific serial device.

The drivers for each physical device can support several serial line protocols. The protocols are divided into two groups, depending on whether they apply to data output by or input to the computer.

Note that the output DTR and input RTS protocols can coexist with other protocols. The strategy is first to set the required character-based protocol and then to set the DTR/RTS protocol. There is an example of this in the following console dialog:

```
P3B>protocol<CR>
PROTOCOL Vn 1.0 02/17/83
PROTOCOL sets the physical device's serial protocols.
        PROTOCOL physical-device direction protocol {message-length}
Legal physical devices are :
                TERMINAL
                PRINTER
                MODEM
Legal direction/protocols are :
                Output DTR
                Output XON
                Output ETX
                Input RTS
                Input XON
        Message length can be specifed with Output ETX.
P3B>protocol show<CR>
PROTOCOL Vn 1.0 02/17/83
        Protocol for TERMINAL - None.
        Protocol for PRINTER - Output XON
        Protocol for MODEM - Input RTS
P3B>protocol <u>m o e 128<CR></u>
PROTOCOL Vn 1.0 02/17/83
        Protocol for TERMINAL - None.
        Protocol for PRINTER - Output XON
```

Protocol for MODEM - Output ETX Message Length 128 bytes.

P3B><u>protocol m o d<CR></u> PROTOCOL Vn 1.0 02/17/83 Protocol for TERMINAL - None. Protocol for PRINTER - Output XON Protocol for MODEM - Output DTR Output ETX Message Length 128 bytes.

```
#define VN "\nPROTOCOL Vn 1.0 02/17/83"
/* PROTOCOL -- This utility sets the serial port protocol for the
specified physical device. Alternatively, it displays the
     current protocols for all of the serial devices. */
#include <LIBRARY.H>
            /* Code tables used to relate ASCII strings to code values */
struct _ct ct_iproto[3]; /* Code table for input protocols */
struct _ct ct_oproto[4]; /* Code table for output protocols */
struct _ct ct_dproto[7]; /* Code table for displaying protocols */
struct _ct ct_dproto[7]; /* Code table for displayin
struct _ct ct_pdev[MAXPDEV + 2];/* Physical device table */
struct _ct ct_io[3]; /* Input, output */
            /* Parameters on the command line *.
#define PDEV argv[1] /* Physical device */
#define IO argv[2] /* Input/output */
#define PROTO argv[3] /* Protocol */
#define PROTOL argv[4] /* Protocol message length */
main(argc,argv)
int arge;
char *argv[];
printf(VN); /* Display sign-on message */
setup(); /* Set up code tables */
chk_use(argc); /* Check correct usage */
           /* Check if request to show current settings */
if (usstrcmp("SHOW",argv[1]))
                                  /* No -- assume a set is required */
           ŧ
           set_proto(get_pdev(PDEV),
                                  dev(PDEV), /* Physical device */
/* Input/output and protocol */
                       get_proto(get_io(IO), PROTO),
                       PROTOL);
                                             /* Protocol message length */
show_proto();
} /* end of program */
                                  /* Set up the code tables for this program */
setup()
ŧ
           /* Initialize the physical device table */
ct_init(ct_pdev[0],0,PN_T);
                                        /* Terminal */
/* Printer */
ct_init(ct_pdev[1],1,PN_P);
ct_init(ct_pdev[2],2,PN_M);
                                             /* Modem */
ct_init(ct_pdev[3],CT_SNF, "*"); /* Terminator */
           /* Initialize the input/output table */
ct_init(ct_io[0],0,"INPUT");
ct_init(ct_io[1],1,"OUTPUT");
ct_init(ct_io[2],CT_SNF,"*");
                                                         /* Terminator */
           /* Initialize the output protocol table */
ct_init(ct_oproto[0],DT_ODTR,"DTR");
ct_init(ct_oproto[1],DT_OXON,"XON");
ct_init(ct_oprotoE2],DT_OETX,"ETX");
```

Figure 11-10. PROTOCOL.C, a utility that sets the protocol governing input and output of a specified serial device

```
ct_init(ct_oproto[3],CT_SNF,"*");
                                                      /* Terminator */
           /* Initialize the input protocol table */
ct_init(ct_iproto[0],DT_IRTS,"RTS");
ct_init(ct_iproto[1],DT_IXON,"XON");
ct_init(ct_iproto[2],CT_SNF,"*");
                                                      /* Terminator */
/* Initialize the display protocol */
ct_init(ct_dproto[0],DT_ODTR,"Output DTR");
ct_init(ct_dproto[1],DT_OZON,"Output TAR");
ct_init(ct_dproto[2],DT_OETX,"Output ETX");
ct_init(ct_dproto[3],DT_IRTS,"Input RTS");
ct_init(ct_dproto[4],DT_IXON,"Input XON");
ct_init(ct_dproto[5],CT_SNF,"*");
unsigned
get_pdev(ppdev)
                               /* Get physical device */
/* This function returns the physical device code
specified by the user in the command line. */
                               /* Pointer to character string */
char *ppdev;
unsigned retval;
                                /* Return value */
retval = ct_parc(ct_pdev,ppdev);/* Get code for ASCII string */
if (retval == CT_SNF) /* If string not found */
          £
           printf("\n\007Physical Device '%s' is invalid or ambiguous.",
           ppdev);
printf("\nLegal Physical Devices are : ");
           ct_disps(ct_pdev);
                                     /* Display all values */
           exit();
           3
return retval:
                                          /* Return code */
unsigned
                              /* Get input/output parameter */
get_io(pio)
char *pio;
                               /* Pointer to character string */
                                           /* Return value */
unsigned retval;
retval = ct_parc(ct_io,pio); /* Get code for ASCII string */
if (retval == CT_SNF) /* If string not found */
           ŧ
           printf("\n\007Input/Output direction '%s' is invalid or ambiguous.",
                     pio);
           printf("\nLegal values are : ");
           ct_disps(ct_io); /* Bisplay all values */
           exit();
           3
                                           /* Return code */
return retval;
unsigned
get_proto(output,pproto)
/* This function returns the protocol code for the
    rotocol specified by the user in the command line. */
t output; /* =1 for output, =0 for input */
ar *pproto; /* Pointer to character string */
int output;
char *pproto;
unsigned retval;
                                           /* Return value */
if (output)
                                           /* OUTPUT specified */
                      /* Get code for ASCII string */
           retval == ct_parc(ct_oproto,pproto);
if (retval == CT_SNF) /* If string not found */
                      printf("\n\0070utput Protocol '%s' is invalid or ambiguous.",
           pproto);
                      printf("\nLegal Output Protocols are : ");
                      ct_disps(ct_oproto); /* Display valid protocols */
                      exit():
                      3
```

Figure 11-10. (Continued)

```
else
                                    /# INPLIT specified #/
         £
         /* Get code for ASCII string */
retval = ct_parc(ct_iproto,pproto);
if (retval == CT_SNF) /* If s
                                            /* If string not found */
                  printf("\n\007Input Protocol '%s' is invalid or ambiguous.",
         pproto);
                  printf("\nLegal Input Protocols are : ");
                  ct_disps(ct_iproto); /* Display valid protocols */
                  exit():
                  3
         3
return retval:
                                    /* Return code */
set_proto(pdevc,protoc,pplength)/* Set the protocol for physical device */
int pdevc:
                                    /* Physical device code */
unsigned protoc;
                                    /* Protocol byte */
char *pplength;
                                    /* Pointer to protocol length */
.
struct _ppdt
char *pdt[16];
                          /* Array of 16 pointers to the device tables */
};
struct _ppdt *ppdt;
struct _dt *dt;
                                    /* Pointer to the device table array */
/* Pointer to a device table */
ppdt = get_cba(CB_DTA); /* Set pointer to array of pointers */
dt = ppdt -> pdt[pdevc];
if (!dt)
                           /* Check if pointer in array is valid */
         £
         printf("\nError -- Array of Device Table Addresses is not set for device #%d.",
                 pdevc);
         exit():
if (protoc & 0x8000)
                          /* Check if protocol byte to be set
                              directly or to be OR ed in */
                           /* OR ed */
         dt -> dt_st1 != (protoc & 0x7F);
else
                           /* Set directly */
         dt -> dt_st1 = (protoc & 0x7F);
if ((protoc & 0x7F) == DT_OETX) /* If ETX/ACK, check for message
                                      length ≭/
         if (isdigit(*pplength))
                                            /* Check if length present */
                  ٤
                           /* Convert length to binary and set device
                              table field. */
                  dt -> dt_etxml = atoi(pplength);
         3
3
show_proto()
                          /* Show the current protocol settings */
struct _ppdt
ş
char *pdt[16];
                          /* Array of 16 pointers to the device tables */
};
struct _ppdt *ppdt;
                                   /* Pointer to the device table array */
                                   /* Pointer to a device table */
/* Physical device code */
struct _dt *dt;
int pdevc;
                                   /* Pointer to display protocols */
struct _ct *dproto;
ppdt = get_cba(CB_DTA); /* Set pointer to array of pointers */
         /* For all physical devices */
```

Figure 11-10. (Continued)

3

```
for (pdevc = 0; pdevc <= MAXPDEV; pdevc++)
        ş
                /* Set pointer to device table */
        dt = ppdt -> pdt[pdevc];
        if (dt) /* Check if pointer in array is valid */
                printf("\n\tProtocol for %s - ",ct_strc(ct_pdev,pdevc));
                        /* Check if any protocols set */
                if (!(dt -> dt st1 & ALLPROTO))
                        £
                        printf("None.");
                        continue;
                        /* Set pointer to display protocol table */
                dproto = ct_dproto;
                while (dproto -> _ct_code != CT_SNF)
                        ŧ
                                /* Check if protocol bit set */
                        /* Move to next entry */
                        ++dproto:
                        /* Check if ETX/ACK protocol and
                message length to be displayed */
if (dt -> dt_st1 & DT_OETX)
printf(" Message length %d bytes.",
                               dt -> dt_etxml);
                3
       1
3
                        /* Check for correct usage */
chk use(argc)
int argc;
                        /* Argument count on commmand line */
if (argc == 1)
        £
        printf("\nPROTOCOL sets the physical device's serial protocols.");
        printf("\n\tPROTOCOL physical-device direction protocol {message-length}");
        printf("\n\nLegal physical devices are :");
        ct_disps(ct_pdev);
        printf("\nLegal direction/protocols are :");
        ct_disps(ct_dproto);
        printf("\n\tMessage length can be specifed with Output ETX.\n");
        exit():
3
```

Figure 11-10. (Continued)

ASSIGN - Assign Physical to Logical Devices

The ASSIGN utility shown in Figure 11-11 sets the necessary bits in the physical input/output redirection bits in the BIOS. It assigns a logical device's input and output to physical devices. Input can only be derived from a single physical device, while output can be directed to multiple devices.

Here is an example console dialog showing ASSIGN in action:

```
P3B>assign<CR>
ASSIGN Vn 1.0 02/17/83
ASSIGN sets the Input/Output redirection.
ASSIGN logical-device INPUT physical-device
ASSIGN logical-device OUTPUT physical-dev1 {phy_dev2..}
ASSIGN SHOW (to show current assignments)
```

Legal logical devices are : CONSOLE AUXILIARY LIST Legal physical devices are : TERMINAL PRINTER MODEM P3B>assign show<CR> ASSIGN Vn 1.0 02/17/83 Current Device Assignments are : CONSOLE INPUT is assigned to - TERMINAL CONSOLE OUTPUT is assigned to - TERMINAL AUXILIARY INPUT is assigned to - MODEM AUXILIARY OUTPUT is assigned to - MODEM LIST INPUT is assigned to - PRINTER LIST OUTPUT is assigned to - PRINTER P3B>assign a o t m p<CR> ASSIGN Vn 1.0 02/17/83 Current Device Assignments are : CONSOLE INPUT is assigned to - TERMINAL CONSOLE OUTPUT is assigned to - TERMINAL AUXILIARY INPUT is assigned to - MODEM AUXILIARY OUTPUT is assigned to - TERMINAL PRINTER MODEM LIST INPUT is assigned to - PRINTER LIST OUTPUT is assigned to - PRINTER

```
#define VN "\nASSIGN Vn 1.0 02/17/83"
#include <LIBRARY.H>
struct _ct ct_pdev[MAXPDEV + 2];
                                          /* Physical device table */
        /* Names of logical devices */
LN_C "CONSOLE"
#define LN_C
#define LN_A
                 "AUXILIARY"
#define LN_L
                 "LIST"
struct _ct ct_ldev[4];
                                  /* Logical device table */
struct _ct ct_io[3];
                                  /* Input, output */
        /* Parameters on the command line */
#define LDEV argv[1] /* Logical device */
#define IO argv[2] /* Input/output */
main(argc,argv)
int argc;
char *argv[];
printf(VN); /* Display sign-on message */
setup():
                 /* Set up code tables */
chk_use(argc); /* Check correct usage */
        /* Check if request to show current settings */
if (usstrcmp("SHOW",argv[1]))
        ŧ
                         /* No, assume a set is required */
```

Figure 11-11. ASSIGN.C, a utility that assigns a logical device's input and output to two physical devices

```
/* NOTE : the number of physical devices to
                        process is given by argc - 3 */
          set_assign(get_ldev(LDEV),get_io(IO),argc - 3,argv);
          7
show_assign();
3
setup()
                              /* Set up the code tables for this program */
          /* Initialize the physical device table */
ct_init(ct_pdev[0],0,PN_T); /* Terminal */
ct_init(ct_pdev[1],1,PN_P); /* Printer */
ct_init(ct_pdev[2],2,PN_M);
                                        /* Modem */
ct_init(ct_pdev[3],CT_SNF,"*"); /* Terminator */
/* Initialize the logical device table */
ct_init(ct_ldev[0],0,LN_C); /* Terminal */
ct_init(ct_ldev[1],LN_A); /* Auxiliary */
ct_init(ct_ldev[2],2,LN_L); /* List */
ct_init(ct_ldev[3],CT_SNF,"*"); /* Terminator */
          /* Initialize the input/output table */
ct_init(ct_io[0],0,"INPUT");
ct_init(ct_io[1],1,"OUTPUT");
ct_init(ct_io[2],CT_SNF,"*");
                                                  /* Terminator */
3
unsigned
get_ldev(pldev) /* Get logical device */
/* This function returns the logical device code
specified by the user in the command line. */
---- *oldev; /* Pointer to character string */
unsigned
unsigned retval;
                                                   /* Return value */
retval = ct_parc(ct_ldev,pldev); /* Get code for A
if (retval == CT_SNF) /* If string not found */
                                                   /* Get code for ASCII string */
          £
          printf("\n\007Logical device '%s' is invalid or ambiguous.",
          pldev);
printf("\nLegal logical devices are : ");
          ct_disps(ct_ldev); /* Display all values */
          exit():
return retval;
                                         /* Return code */
unsigned
                            /* Get input/output parameter */
get_io(pio)
char *pio;
                              /* Pointer to character string */
                                         /* Return value */
unsigned retval;
retval = ct_parc(ct_io,pio); /* Get code for ASCII string */
if (retval == CT_SNF) /* If string not found */
if (retval == CT_SNF)
          £
          printf("\n\007Input/output direction '%s' is invalid or ambiguous.",
          pio);
printf("\nLegal values are : ");
          ct_disps(ct_io);
                                       /* Display all values */
          exit();
          ł
return retval;
                                         /* Return code */
/* Set assignment (I/O redirection) */
                                        /* Logical device code */
                                         /* I/O redirection code */
int output;
                                         /* count of arguments to process */
/* Replica of parameter to main function */
int argc;
char *argv[];
unsigned *redir:
                                         /* Pointer to redirection word */
                                         /* Physical device code */
int pdevc;
                                         /* Redirection value */
unsigned rd_val;
          /* Get the address of the I/O redirection word.
```

Figure 11-11. (Continued)

```
This code assumes that get_cba code values
              are ordered:
                    Device #0, input & output
                    Device #1, input & output
                    Device #2, input & putput
             The get_cba code is computed by multiplying the logical device code by 2 (that is, shift left 1) and added onto the code for Device #0, input Then the output variable (0 = input, 1 = output) is added on */
redir = get_cba(CB_CI + (ldevc << 1) + output);</pre>
rd val = 0:
                    /* Initialize redirection value */
          /* For output, assignment can be made to several physical
             devices, so this code may be executed several times
do
          £
                    /* Get code for ASCII string */
                    /* NOTE: the physical device parameters start
with parameter #3 (argv[3]). However argc
is a decreasing count of the number of physical
                        devices to be processed, Therefore, argc + 2
                        causes them to be processed in reverse order
                        (i.e. from right to left on the command line) */
          pdevc = ct_parc(ct_pdev,argv[argc + 2]);
          if (pdevc == CT_SNF)
                                                  /* If string not found */
                    printf("\n\007Physical device '%s' is invalid or ambiguous.",
                    argv[argc + 2]);
                    printf("\nLegal physical devices are : ");
                    ct_disps(ct_pdev);
                                                 /* Display all values */
                    exit():
                    /* Repeat this loop for as long as there are
more parameters (for output only) */
          else
                    /* Build new redirection value by OR ing in
                    a one-bit shifted left pdevc places. */
rd_val != (1 << pdevc);</pre>
          3 while (--argc && output);
                            /* Set the value into the config. block */
*redir = rd val;
show assign()
                                       /* Show current baud rate */
int rd_code;
                                        /* Redirection code for get_cba */
int ldevn;
                                        /* Logical device number */
int pdevn;
                                        /* Physical device number */
unsigned rd_val;
                                        /* Redirection value */
unsigned *prd_val;
                                        /* Pointer to the redirection value */
/* Note: the respective codes for accessing the redirection values
via the get_cba (get configuration block address) function are:
         Device #0 console input -- 5
Device #0 console putput -- 6
         Device #1 auxiliary input -- 7
Device #1 auxiliary output -- 8
Device #2 list input -- 9
          Device #2 list output -- 10
   This function uses this mathematical relationship */
printf("\nCurrent device assignments are :");
          /* For all get_cba codes */
for (rd_code = CB_CI; rd_code <= CB_L0; rd_code++)</pre>
                   /* Set pointer to redirection value */
```

Figure 11-11. (Continued)

```
rd_val = *prd_val;
                                      /* This also performs byte reversal */
                   /* Display device name. The rd_code is converted to a
                      device number by subtracting the first code number
                       from it and dividing by 2 (shift right one place).
                       The input/output direction is derived from the
                       least significant bit of the rd_code. */
         /* For all physical devices */
         for (pdevn = 0; pdevn < 16; pdevn++)
                             /* Check if current physical device is assigned
                   by AND ing with a 1-bit shifted left pdevn times */
if (rd_val & (1 << pdevn)) /* Is device active? */
                                     /* Display physical device name */
                            printf(" %s",ct_strc(ct_pdev,pdevn) );
                   ł
         3
æ
                            /* Check for correct usage */
chk_use(argc)
int_argc;
                            /* Argument count on commmand line */
£
if (argc == 1)
         8
         t
printf("\nASSIGN sets the Input/Output redirection.");
printf("\n\tASSIGN logical-device INPUT physical-device");
printf("\n\tASSIGN logical-device OUTPUT physical-dev1 {phy_dev2..}");
printf("\n\tASSIGN SHOW (to show current assignments)");
         printf("\n\nLegal logical devices are :");
         ct_disps(ct_ldev);
         printf("\nLegal physical devices are :");
         ct_disps(ct_pdev);
         exit();
3
```

Figure 11-11. (Continued)

DATE — Set the System Date

The DATE utility shown in Figure 11-12 sets the system date in the configuration block, along with a flag that indicates that the DATE utility has been used. Other utility programs can use this flag as a primitive test of whether the system date is current.

Here is an example console dialog:

```
P3B><u>date<CR></u>
DATE Vn 1.0 02/18/83
DATE sets the system date. Usage is :
DATE mm/dd/yy
DATE SHOW (to display current date)
P3B><u>date show<CR></u>
DATE Vn 1.0 02/18/83
Current Date is 12/18/82
P3B><u>date 2/23/83<CR></u>
DATE Vn 1.0 02/18/83
Current Date is 02/23/83
```

```
#define VN "\nDATE Vn 1.0 02/18/83"
/* This utility accepts the current date from the command tail,
    validates it, and set the internal system date in the BIOS.
   Alternatively, it can be requested just to display the current system date. */
#include <LIBRARY.H>
                           /* Pointer to the date in the config. block */
/* Pointer to date-set flag */
char *date:
char #date_flag;
int mm,dd,yy;
                           /* Variables to hold month, day, year */
/* Match count of numeric values entered */
int mcount;
int count;
                            /* Count used to add leading O's to date */
main(argc,argv)
int arge;
char *argv[];
£
printf(VN); /* Display sign-on message */
date = get_cba(CB_DATE); /* Set pointer to date */
date_flag = get_cba(CB_DTFLAGS);/* Set pointer to date-set flag */
if (argc != 2)
                            /* Check if help requested (or needed) */
         show_use();
                           /* Display correct usage and exit */
if (usstrcmp("SHOW",argv[1])) /* Check if not SHOW option */
         £
         /* Convert specified time into month, day, year */
mcount = sscanf(argv[1], "%d/%d/%d",%mm,&dd,&yy);
if (mcount != 3)
/* Input not numeric */
                                    /* Display correct usage and exit */
                   show use():
                   /* NOTE: The following validity checking is
                      simplistic, but could be expanded to accommodate
                      more context-sensitive checking: days in the month,
         leap years, etc. */
if (mm > 12 !! mm < 1) /* Check valid month, day, year */</pre>
                   printf("\nMonth = %d is illegal.",mm);
                   show_use();
                                    /* Display correct usage and exit */
                   3
         if (dd > 31 + dd < 1)
                   ٤.
                   if (yy > 90 || yy < 83) /* <=== NOTE ! */
                   £
                   printf("\nYear = %d is illegal.", yy);
                   show_use(); /* Display correct usage and exit */
         /* Convert integers back into a formatted string */
sprintf(date,"%2d/%2d/%2d",mm,dd,yy);
         date[8] = 0x0A; /* Terminate with line feed */
date[9] = '\0'; /* New string terminator */
                   /* Change " 1/ 2/ 3" into "01/02/03" */
         for (count = 0; count \langle 7; count+=3 \rangle
                   £
                   if (date[count] == < <)
                           date[count] = '0';
                   3
                   /* Turn flag on to indicate that user has set date */
         *date_flag != DATE_SET;
printf("\n\tCurrent Date is %s",date);
                            /* Display correct usage and exit */
show_use()
printf("\nDATE sets the system date. Usage is :");
printf("\n\tDATE mm/dd/yy");
printf("\n\tDATE SHOW (to display current date)\n");
exit();
ł
```

Figure 11-12. DATE.C, a utility that makes the current date part of the system

TIME — Set the System Time

The TIME utility shown in Figure 11-13 sets the current system time. Like DATE, TIME sets a flag so that other utilities can test that the system time is likely to be current.

Here is an example console dialog:

P3B><u>time<CR></u> TIME Vn 1.0 02/18/83 TIME sets the system time. Usage is : TIME hh{:mm{:ss}} TIME SHOW (to display current time)

P3B><u>time show<CR></u> TIME Vn 1.0 02/18/83 Current Time is 13:08:44

P3B><u>time 5:47<CR></u> TIME Vn 1.0 02/18/83 Current Time is 05:47:00

```
#define VN "\nTIME Vn 1.0 02/18/83"
/* This utility accepts the current time from the command tail,
validates it, and sets the internal system time in the BIOS.
   Alternatively, it can just display the current system time. */
#include <LIBRARY.H>
                         /* Pointer to the time in the config. block */
/* Pointer to the time set flag */
char *time;
char *time_set;
int hh,mm,ss;
                          /* Variables to hold hours, minutes, seconds */
int mcount;
                          /* Match count of numeric values entered */
int count;
                          /* Count used to add leading zeros to time */
main(argc.argv)
int argc:
char *argv[];
printf(VN);
                          /* Display sign-on message */
time = get_cba(CB_TIMEA); /* Set pointer to time */
time = get_cba(CB_IIMEA); /* Set Pointer to the
time_flag = get_cba(CB_DTFLAGS); /* Set Pointer to the
time-set flag */
                          /* Initialize the time if seconds or
hh = mm = ss = 0;
                             minutes are not specified */
                          /* Check if help requested (or needed) */
if (argc != 2)
         show_use(); /* Display correct usage and exit */
if (usstrcmp("SHOW",argv[1]))
                                   /* Check if not SHOW option */
         f
                  /* Convert time into hours, minutes, seconds */
         mcount = sscanf(argv[1], "%d:%d:%d",&hh,&mm,&ss);
        if (!meount)
                                  /* Input not numeric */
                 show_use();
                                  /* Display correct usage and exit */
         if (hh > 12)
                                   /* Check valid hours. minutes. seconds */
                  £
                  printf("\n\007Hours = %d is illegal.",hh);
                                   /* Display correct usage and exit */
                  show_use();
```

Figure 11-13. TIME.C, a utility that makes the current time part of the system

```
if (mm > 59)
                £
                printf("\n\007Minutes = %d is illegal.",mm);
                show_use();
                              /* Display correct usage and exit */
        if (ss > 59)
                ş
               show_use(); /* Display correct usage and exit */
printf("\n\007Seconds = %d is illegal.",ss);
                /* Convert integers back into formatted string */
        /* Convert " 1: 2: 3" into "01:02:03" */
        for (count = 0; count < 7; count+=3)
                if (time[count] == / /)
                       time[count] = '0';
                3
                /* Turn bit on to indicate that the time has been set */
        *time_flag != TIME_SET;
printf("\n\tCurrent Time is %s",time);
show_use()
                       /* Display correct usage and exit */
printf("\nTIME sets the system time. Usage is :");
printf("\n\tTIME hh{:mm{:ss}}")
printf("\n\tTIME SHOW (to display current time)\n");
exit():
3
```

Figure 11-13. TIME.C, a utility that makes the current time part of the system (continued)

FUNKEY — Set the Function Keys

The FUNKEY utility shown in Figure 11-14 sets the character strings associated with specific function keys. In the specified character string, the character "<" is converted into a LINE FEED character. Here is an example console dialog:

```
P3B>funkey<CR>
FUNKEY sets a specific function key string.
        FUNKEY key-number "string to be programmed<"
                   (Note : '<' is changed to line feed.)
                          key-number is from 0 to 17.)
                   (
                   (
                           string can be up to 16 chars.)
        FUNKEY SHOW
                          (displays settings for all keys)
P3B>funkey show<CR>
FUNKEY Vn 1.0 02/18/83
        Key #0 = 'Function Key 1<'
        Key #1 = 'Function Key 2<'
P3B>funkey 0 "PIP B:=A:*.*[V]<"<CR>
P3B>funkey show<CR>
FUNKEY Vn 1.0 02/18/83
       Key #0 = 'PIP B:=A:*.*[V]<'
        Key #1 = 'Function Key 2<'
```

```
#define VN "\nFUNKEY Vn 1.0 02/18/83"
#include <LIBRARY.H>
int fnum:
                                   /* Function key number to be programmed */
                                   /* String for function key */
/* Pointer to function key table */
char fstring[20];
struct _fkt *pfk;
main(argc,argv)
int argc;
char *argv[];
if (argc == 1 || argc > 3)
         show_use();
pfk = get_cba(CB_FKT); /* Set pointer to function key table */
if (usstremp("SHOW",argv[1]))
         if (!isdigit(argv[1][0]))
                  printf("\n\007'%s' is an illegal function key.",
                           argv[1]);
                  show_use();
         fnum = atoi(argv[1]); /* Convert function key number */
         if (fnum > FK_ENTRIES)
                  printf("\n\007Function key number %d too large.",fnum);
                  show_use();
         if (get_fs(fstring) > FK_LENGTH)
                  printf("\n\007Function key string is too long.");
                  show_use();
        pfk += fnum:
                          /* Update pointer to string */
                  /* Copy string into function key table */
         /* Check if function key input present */
if (!(pfk -> fk_input[0]))
                  £
                  printf("\n\007Error : Function Key #%d is not set up to be programmed.",fnum);
                  show_use();
         strcpy(pfk -> fk_output,fstring);
else
                  /* SHOW function specified */
         £
         printf(VN);
                                    /* Display sign-on message */
         show_fun();
3
                           /* Get function string from command tail */
/* Pointer to character string */
get_fs(string)
char string[];
£
                           /* Pointer to command tail */
/* Count of TOTAL characters in command tail */
/* String length */
char *tail:
short tcount;
int slen;
tail = 0x80;
tcount = *tail++;
                           /* Command line is in memory at 0080H */
/* Set TOTAL count of characters in command tail */
slen = O;
                           /* Initialize string length */
                           /* For all characters in the command tail */
while(tcount--)
         if (*tail++ == '"')
                                   /* Scan for first quotes */
                  break;
```

Figure 11-14. FUNKEY.C, a utility that sets the character strings associated with specific function keys

```
3
 if (!tcount)
                           /* No quotes found */
          Ŧ
          printf("\n\007No leading quotes found.");
          show_use();
                            /* Adjust tail count */
 ++tcount;
                           /* For all remaining characters in tail */
 while(tcount--)
          .
          if (*tail == '"')
                   £
                   string[slen] = '\0':
                                             /* Add terminator */
                   break; /* Exit from loop */
          string[slen] = *tail++; /* Move char. from tail into string */
          if (string[slen] == '<')
                 string[slen] = 0x0A;
          ++slen;
          3
if (!tcount)
                            /* No terminating quotes found */
          ŧ
         printf("\n\007No trailing quotes found.");
          show_use();
return slen;
                           /* Return string length */
                           /* Display settings for all function keys */
show_fun()
 struct _fkt *pfkt;
                            /* Local pointer to function keys */
                           /* Count to access function keys */
/* Pointer to "<" character (LINE FEED) */
int count;
char *1f;
pfkt = get_cba(CB_FKT); /* Set pointer to function key table */
for (count = 0; count <= FK_ENTRIES; count++)
         Ŧ
         if (pfkt -> fk_input[0])
                                              /* Key is programmed */
                  £
                           /* Check if at physical end of table */
                  if (pfkt -> fk_input == 0xFF)
    break; /* Yes -- break out of for loop */
strcpy(fstring,pfkt -> fk_output);
                  /* Convert all 0x0A chars to "<" */
while (lf = strscn(fstring,"\012"))</pre>
                            £
                           *1f = '<';
                            3
                  printf("\n\tKey #%d = '%s'",count,fstring);
                  3
         ++pfkt;
                           /* Move to next entry */
3
show_use()
printf("\nFUNKEY sets a specific function key string.");
print((\n\tFUNKEY key-number \042string to be programmed(\042 ");
printf("\n\t (Note : << is changed to line feed.)");
printf("\n\t
                         (
                                  key-number is from 0 to %d.)",
FK_ENTRIES-1);
printf("\n\t
                         (
                                  string can be up to %d chars.)",
FK_LENGTH);
printf("\n\tFUNKEY SHOW
                                 (displays settings for all keys)");
exit():
```

Figure 11-14. (Continued)

Other Utilities

Because of space limitations, not all of the possible utility programs for the BIOS features can be shown in this chapter. Others that would need to be developed in order to have a complete set are

PUBLIC/PRIVATE

This pair of utilities would turn the public files flag on or off, making the files in user 0 available from other user numbers or not, respectively.

SETTERM

This program would program the CONOUT escape table, setting the various escape sequences as required. It could also program the characters in the function key table that match with those emitted by the terminal currently in use.

SAVESYS

This utility would save the current settings in the long term configuration block.

LOADSYS

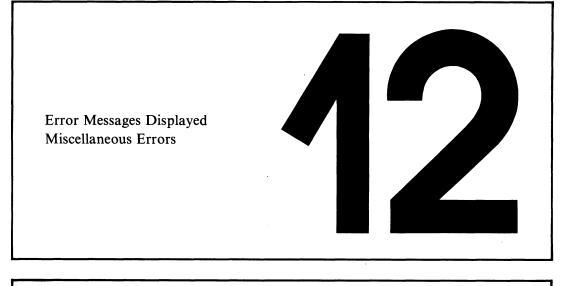
This would load the long term configuration block from a previously saved image.

DO

This utility would copy the command tail into the multi-command buffer, changing "\" into LINE FEED, and then set the forced input pointer to the multi-command buffer. As a result, characters from the multi-command buffer would be fed into the console input stream as though they had been typed one command at a time.

SPARE

This utility would work in conjunction with the hard-disk bad-sector management in your disk drivers. It would spare out bad sectors or tracks on the hard disk. This done, all subsequent references to the sectors or tracks would be redirected to a different part of the disk.



Error Messages

This chapter lists the error messages that emanate from standard CP/M and its utility programs. It does not include any error messages from the BIOS; these messages, if any, are the individualized product of the programmers who wrote the various versions of the BIOS.

The error messages are shown in alphabetical order, followed (in parentheses) by the name of the program or CP/M component outputting the message. Messages are shown in uppercase even if the actual message you will see contains lowercase letters. Additional characters that are displayed to "pretty up" the message have been omitted. For example, the message "** ABORTED **" will be listed as "ABORTED".

Following each message is an explanation and, where possible, some information to help you deal with the error.

The last section of the chapter deals with known errors or peculiarities in CP/M and its utilities. Read this section so that you will recognize these problems when they occur.

Error Messages Displayed

? (CCP)

The CCP displays a question mark if you enter a command name and there is no corresponding "command.COM" file on the disk.

It is also displayed if you omit the number of pages required as a parameter in the SAVE command.

? (DDT)

DDT outputs a question mark under several circumstances. You must use context (and some guesswork) to determine what has gone wrong. Here are some specific causes of problems:

- DDT cannot find the file that you have asked it to load into memory. Exit from DDT and investigate using DIR or STAT (the file may be set to System status and therefore invisible with DIR).
- There is a problem with the data in the HEX file that you have asked DDT to load. The problem could be a bad check-sum on a given line or an invalid field somewhere in the record. Try typing the HEX file out on a console, or use an editor to examine it. It is rare to have only one or two bad bits or bytes in a HEX file; large amounts of the file are more likely to have been corrupted. Therefore, you may be able to spot the trouble fairly readily. If you have the source code for the program, reassemble it to produce another copy of the HEX file. If you do not have the source code, there is no reliable way around this problem unless you are prepared to hand-create the HEX file—a difficult and tedious task.
- DDT does not recognize the instruction you have entered when using the "A" (assemble) command to convert a source code instruction into hexadecimal. Check the line that you entered. DDT does not like tabs in the line (although it appears to accept them) or hexadecimal numbers followed by "H". Check that the mnemonic and operands are valid, too.

??= (DDT)

This cryptic notation is used by DDT when you are using the "L" (list disassembled) command to display some part of memory in DDT's primitive assembly language form. DDT cannot translate all of the 256 possible values of a byte. Some of them are not used in the 8080 instruction set. When DDT encounters an untranslatable value, it displays this message as the instruction code, followed by the actual value of the byte in hexadecimal.

You will see this if you try to disassemble code written for the Z80 CPU, which

uses unassigned 8080 instructions. You will also see it if you try to disassemble bytes that contain ASCII text strings rather than 8080 instructions.

ABORTED (STAT)

If you enter any keyboard character while STAT is working its way down the file directory setting files to DIR (Directory), SYS (System), R/W (Read/Write), or R/O (Read-Only) status, then it will display this message, stop what it is doing, and execute a warm boot.

By contrast, if you enter the command

A><u>stat</u> <u>*.*<cr></u>

to display all of the files on a disk, there is no way that the process can be aborted.

ABORTED (PIP)

This message is displayed if you press any keyboard character while PIP is copying a file to the list device.

BAD DELIMITER (STAT)

If your BIOS uses the normal IOBYTE method of assigning physical devices to logical devices, you use STAT to perform the assignment. The command has this format:

STAT RDR:=PTR:

STAT displays this message if it cannot find the "=" in the correct place.

BAD LOAD (CCP)

This is probably the most obscure error message that emanates from CP/M. You will get this message if you attempt to load a COM file that is larger than the transient program area. Your only recourse is to build a CP/M system that has a larger TPA.

BAD PARAMETER (PIP)

PIP accepts certain parameters in square brackets at the end of the command line. This message is displayed if you enter an invalid parameter or an illegal numeric value following a parameter letter.

BDOS ERROR ON d: BAD SECTOR (BDOS)

The BDOS displays this message if the READ and WRITE functions in your BIOS ever return indicating an error. The only safe response to this message is to type CONTROL-C. CP/M will then execute a warm boot. If you type CARRIAGE RETURN, the error will be ignored—with unpredictable results.

A well-implemented BIOS should include disk error recovery and control so that the error will never be communicated to the BDOS. If the BIOS gives you the option of ignoring an error, do so only when you are reasonably sure of the outcome or have adequate backup copies so that you can recreate your files.

BDOS ERROR ON d: FILE R/O (BDOS)

You will see this message if you attempt to erase (ERA) a file that has been set to Read-Only status. Typing any character on the keyboard causes the BDOS to perform a warm boot operation. Note that the BDOS does not tell you *which* file is creating the problem. This can be a problem when you use ambiguous file names in the ERA command. Use the STAT command to display all the files on the disk; it will tell you which files are Read-Only.

This message is also displayed if a program tries to delete a Read-Only file. Again, it can be difficult to determine which file is causing the problem. Your only recourse is to use STAT to try to infer which of the Read-Only files might be causing the problems.

BDOS ERROR ON d: R/O (BDOS)

This looks similar to the previous message, but it refers to an entire logical disk instead of a Read-Only file. However, it is rarely output because you have declared a disk to be Read-Only. Usually, it occurs because you changed diskettes without typing a CONTROL-C; CP/M will detect the new diskette and, without any external indication, will set the disk to Read-Only status.

If you or a program attempts to write any data to the disk, the attempt will be trapped by the BDOS and this message displayed. Typing any character on the keyboard causes a warm boot—then you can proceed.

BDOS ERROR ON d: SELECT (BDOS)

The BDOS displays this message if you or a program attempts to select a logical disk for which the BIOS lacks the necessary tables. The BDOS uses the value returned by SELDSK to determine whether a logical disk "exists" or not.

If you were trying to change the default disk to a nonexistent one, you will have to press the RESET button on your computer. There is no way out of this error.

However, if you were trying to execute a command that accessed the nonexistent disk, then you can type a CONTROL-C and CP/M will perform a warm boot.

BREAK x AT y (ED)

This is another cryptic message whose meaning you cannot guess. The list that follows explains the possible values of "x." The value "y" refers to the command ED was executing when the error occurred.

Meaning

- # Search failure. ED did not find the string you asked it to search for.
- ? Unrecognized command.
- 0 File not found.
- > ED's internal buffer is full.
- E Command aborted.
- F Disk or directory full. You will have to determine which is causing the problem.

CANNOT CLOSE, READ/ONLY? (SUBMIT)

x

SUBMIT displays this message if the disk on which it is trying to write its output file, "\$\$\$.SUB", is physically write protected. Do not confuse this with the disk being *logically* write protected.

The standard version of SUBMIT writes the output file onto the current default disk, so if your current default disk is other than drive A:, you may be able to avoid this problem if you switch the default to A: and then enter a command of the form

A><u>submit</u> <u>b:subfile<cr></u>

CANNOT CLOSE DESTINATION FILE (PIP)

PIP displays this message if the destination disk is physically write protected. Check the destination disk. If it is write protected, remove the protection and repeat the operation.

If the disk is not protected, you have a hardware problem. The directory data written to the disk is being written to the wrong place, even the wrong disk, or is not being recorded on the medium.

CANNOT CLOSE FILES (ASM)

ASM displays this message if it cannot close its output files because the disk is physically write protected, or if there is a hardware problem that prevents data being written to the disk. See the paragraph above.

CANNOT READ (PIP)

PIP displays this message if you attempt to read information from a logical device that can only output. For example:

A>pip diskfile=LST:<<r>

PIP also will display this message if you confuse it sufficiently, as with the following instruction:

A>pip file1=file2;file3<cr>

CANNOT WRITE (PIP)

PIP displays this message if you attempt to output (write) information to a logical device that can only be used for input, such as the RDR: (reader, the anachronistic name for the auxiliary input device).

CHECKSUM ERROR (LOAD)

LOAD displays this message if it encounters a line in the input HEX file that does not have the correct check sum for the data on the line.

LOAD also displays information helpful in pinpointing the problem:

```
CHECKSUM ERROR
LOAD ADDRESS 0110 <- First address on line in file
ERROR ADDRESS 0112 <- Address of next byte to be loaded
BYTES READ:
0110:
010: 00 33 22 2B 02 21 27 02 <- Bytes preceding error
```

Note that LOAD does not display the check-sum value itself. Use TYPE or an editor to inspect the HEX file in order to see exactly what has gone wrong.

CHECKSUM ERROR (PIP)

If you ask PIP to copy a file of type HEX, it will check each line in the file, making sure that the line's check sum is valid. If it is not, PIP will display this message. Unfortunately, PIP does not tell you which line is in error—you must determine this by inspection or recreate the HEX file and try again.

COMMAND BUFFER OVERFLOW (SUBMIT)

SUBMIT displays this message if the SUB file you specified is too large to be processed. SUBMIT's internal buffer is only 2048 bytes. You must reduce the size of the SUB file; remove any comment lines, or split it into two files with the last line of the first file submitting the second to give a nested SUBMIT file.

COMMAND TOO LONG (SUBMIT)

The longest command line that SUBMIT can process is 125 characters. There is no way around this error other than reducing the length of the offending line. You will have to find this line by inspection—SUBMIT does not identify the line.

One way that you can remove a few characters from a command line is to rename the COM file you are invoking to a shorter name, or use abbreviated names for parameters if the program will accept these.

CORRECT ERROR, TYPE RETURN OR CTL-Z (PIP)

This message is a carryover from the days when PIP used to read hexadecimal data from a high-speed paper tape reader. If PIP detected the end of a physical roll

of paper tape, it would display this message. The user could then check to see if the paper tape had torn or had really reached its end. If there was more tape to be read, the user could enter a CARRIAGE RETURN to resume reading tape or enter a CONTROL-Z to serve as the end-of-file character.

Needless to say, it is unlikely that you will see this message if you do not have a paper tape reader.

DESTINATION IS R/O, DELETE (Y/N)? (PIP)

PIP displays this message if you try to overwrite a disk file that has been set to Read-Only status. If you type "Y" or "y", PIP will overwrite the destination file. It leaves the destination file in Read/Write status with its Directory/System status unchanged. Typing any character other than "Y" or "y" makes PIP abandon the copy and display the message

** NOT DELETED**

You can avoid this message altogether if you specify the "w" option on PIP's command line. For example:

```
A>pip destfile=srcfile[w]<cr>
```

PIP will then overwrite Read-Only files without question.

DIRECTORY FULL (SUBMIT)

This message is displayed if the BDOS returns an error when SUBMIT tries to create its output file, "\$\$\$.SUB". As a rough and ready approximation, use "STAT *.*" to see how many files and extents you have on the disk. Erase any unwanted ones. Then use "STAT DSK:" to find out the maximum number of directory entries possible for the disk.

You may also see this message if the file directory has become corrupted or if the disk formatting routine leaves the disk with the file directory full of some pattern other than E5H.

You can assess whether the directory has been corrupted by using "STAT USR:". STAT then displays which user numbers contain files. If the directory is corrupt, you will normally see user numbers greater than 15.

It is not easy to repair a corrupted directory. "ERA *.*" erases only the files for the current user number, so you will have to enter the command 16 times, once for each user number from 0 to 15. Alternatively, you can reform the disk.

DISK OR DIRECTORY FULL (ED)

Self-explanatory.

DISK READ ERROR (PIP) DISK WRITE ERROR (SUBMIT) DISK WRITE ERROR (PIP)

These messages will normally be preceded by a BIOS error message. They will only be displayed if the BIOS returns indicating an error. As was described earlier, this is unlikely if the BIOS has any kind of error recovery logic.

END OF FILE, CTL-Z? (PIP)

PIP displays this message if, while copying a HEX file, it encounters a CONTROL-Z (end of file). Again, the underlying idea is based on the concept of physical paper tape. When you saw this message, you could look at the tape in the reader, and if it really was at the end of the roll, enter a CONTROL-Z on the keyboard to terminate the file. Given any other character, PIP would read the next piece of tape.

ERROR : CANNOT CLOSE FILES (LOAD)

LOAD displays this message if you have physically write protected the disk on which it is trying to write the output COM file.

ERROR : CANNOT OPEN SOURCE (LOAD)

LOAD displays this message if it cannot open the HEX file that you specified in the command tail.

ERROR : DISK READ (LOAD) ERROR : DISK WRITE (LOAD)

These two messages would normally be preceded by a BIOS error message. If your BIOS includes disk error recovery, you would not normally see these messages; the error would have been handled by the BIOS.

ERROR : INVERTED LOAD ADDRESS (LOAD)

LOAD displays this message if it detects a load address less than 0100H in the input HEX file. It also displays the actual address input from the file, so you can examine the HEX file looking for this address to determine the likely cause of the problem.

Note that DDT, when asked to load the same HEX file, will do so without any error—and will probably damage the contents of the base page in so doing.

ERROR : NO MORE DIRECTORY SPACE (LOAD)

Self-explanatory.

ERROR ON LINE N (SUBMIT)

SUBMIT displays this message if it encounters a line in the SUB file that it does not know how to process. Most likely you have a file that has type .SUB but does not contain ASCII text.

The first line of the SUB file is number 001.

FILE EXISTS (CCP)

The CCP displays this message if you attempt to use the REN command to rename an existing file to a name already given to another file.

Use "STAT *.*" to display all of the files on the disk. DIR will show only those files that have Directory status, and you may not be able to see the file causing the problem.

FILE IS READ/ONLY (ED)

ED displays this message if you attempt to edit a file that has been set to Read-Only status.

FILE NOT FOUND (STAT) FILENAME NOT FOUND (PIP)

STAT and PIP display their respective messages if you specify a nonexistent file. This applies to both specific and ambiguous file names.

INVALID ASSIGNMENT (STAT)

STAT can be used to assign physical devices to logical devices using the IOBYTE system described earlier. It will display this message if you enter an illogical assignment. Use the "STAT VAL:" command to display the valid assignments.

INVALID CONTROL CHARACTER (SUBMIT)

SUBMIT is supposed to be able to handle a control character in the SUB file—the notation being " x ", where "x" is the control letter. In fact, the standard release version of SUBMIT cannot handle this notation. A patch is available from Digital Research to correct this problem.

Given that this patch has been installed, SUBMIT will display this message if a character other than "A" to "Z" is specified after the circumflex character.

INVALID DIGIT (PIP)

PIP displays this message if it encounters non-numeric data where it expects a numeric value.

INVALID DISK ASSIGNMENT (STAT)

STAT displays this message if you try to set a logical disk to Read-Only status and you specify a parameter other than "R/O." Note that there is no leading "\$" in this case (as there is when you want to set a file to Read-Only).

INVALID DRIVE NAME (USE A, B, C, OR D) (SYSGEN)

SYSGEN displays this message if you attempt to load the CP/M system from, or write the system to, a disk drive other than A, B, C, or D.

INVALID FILE INDICATOR (STAT)

STAT outputs this message if you specify an erroneous file attribute. File attributes can only be one of the following:

\$DIR	Directory
\$SYS	System
\$R/O	Read-Only
\$R/W	Read/Write

INVALID FORMAT (PIP)

PIP displays this message if you enter a badly formatted command; for example, a "+" character instead of an "=" (on some terminals these are on the same key).

INVALID HEX DIGIT (LOAD)

LOAD displays this message if it encounters a nonhexadecimal digit in the input HEX file, where only a hex digit can appear. LOAD then displays additional information to tell you where in the file the problem occurred:

```
INVALID HEX DIGIT
LOAD ADDRESS 0110 <- First address on line in file
ERROR ADDRESS 0112 <- Address of byte containing non-hex
BYTES READ:
0110:
0110: 00 33 <- Bytes preceding error
```

INVALID MEMORY SIZE (MOVCPM)

MOVCPM displays this message if you enter an invalid memory size for the CP/M system size you want to construct.

INVALID SEPARATOR (PIP)

PIP displays this message if you try to concatenate files using something other than a comma between file names.

INVALID USER NUMBER (PIP)

PIP displays this message if you enter a user number outside the range 0 to 15 with the "[gn]" option (where "n" is the user number).

NO 'SUB' FILE PRESENT (SUBMIT)

SUBMIT displays this message if it cannot find a file with the file name that you specified and with a type of .SUB.

NO DIRECTORY SPACE (ASM) NO DIRECTORY SPACE (PIP)

Self-explanatory.

NO FILE (CCP)

The CCP displays this message if you use the REN (rename) command and it cannot find the file you wish to rename.

NO FILE (PIP)

PIP displays this message if it cannot find the file that you specified.

NO MEMORY (ED)

ED displays this message if it runs out of memory to use for storing the text that you are editing.

NO SOURCE FILE ON DISK (SYSGEN)

This error message is misleading. SYSGEN does not read source code files. The message should read "INPUT FILE NOT FOUND".

NO SOURCE FILE PRESENT (ASM)

In this case, ASM really does mean that the source code file cannot be found. Remember that ASM uses a strange form of specifying its parameters. ASM uses the file name that you enter and then searches for a file of that name, but with file type .ASM. The three characters of the file type that you specify are used to represent the logical disks on which the source, hex, and list files, respectively, are to be placed.

NO SPACE (CCP)

The CCP displays this message if you use the SAVE command and there is insufficient room on the disk to accommodate the file.

NOT A CHARACTER SOURCE (PIP)

PIP displays this message if you attempt to copy characters from a character output device, such as the auxiliary output device (known to PIP as PUN:).

OUTPUT FILE WRITE ERROR (ASM)

ASM will display this message if the BDOS returns an error from a disk write operation. If your BIOS has disk error recovery logic, you should never see this message.

PARAMETER ERROR (SUBMIT)

SUBMIT uses the "\$" to mark points where parameter values are to be substituted. If you have a single "\$" followed by an alphabetic character, SUBMIT will display this message. Use "\$\$" to represent a real "\$".

PERMANENT ERROR, TYPE RETURN TO IGNORE (SYSGEN)

SYSGEN displays this message if the BIOS returns an error from a disk read or write operation. If your BIOS has disk error recovery logic, you should never see this message.

QUIT NOT FOUND (PIP)

PIP displays this message when it cannot find the string specified in the "[Qcharacter string^AZ]" option, meaning "Quit copying when you encounter this string."

READ ERROR (CCP)

The CCP displays this message if the BIOS returns an error from a disk read or write operation. If your BIOS includes disk error recovery logic, you should not see this error message.

RECORD TOO LONG (PIP)

PIP displays this message if it encounters a line longer than 80 characters while copying a HEX file. Inspect the HEX file using the TYPE command or an editor.

REQUIRES CP/M 2.0 OR NEWER FOR OPERATION (PIP) REQUIRES CP/M VERSION 2.0 OR LATER (XSUB)

Self-explanatory.

SOURCE FILE INCOMPLETE (SYSGEN)

SYSGEN displays this message if the file that you have asked it to read is too short. Use STAT to check the length of the file.

SOURCE FILE NAME ERROR (ASM)

ASM displays this message if you specify an ambiguous file name: that is, one that contains either "*" or "?".

SOURCE FILE READ ERROR (ASM)

ASM displays this message if it encounters problems reading the input source code file. Check the input file using the TYPE command or an editor.

START NOT FOUND (PIP)

PIP displays this message when it cannot find the string specified in the "[Scharacter stringZ]" option, meaning "Start copying when you encounter this string."

SYMBOL TABLE OVERFLOW (ASM)

ASM displays this message when you have too many symbols in the source code file. Your only recourse is to split the source file into several pieces and arrange for ORG (origin) statements to position the generated object code so that the pieces fit together.

SYNCRONIZATION ERROR (MOVCPM)

Apart from the spelling error, this message is designed to be cryptic. MOVCPM displays it when the Digital Research serial number embedded in MOVCPM does not match the serial number in the version of CP/M that you are currently running.

SYSTEM FILE NOT ACCESSIBLE (ED)

ED displays this message if you attempt to edit a file that has been set to System status. Use STAT to set the file to Directory status.

TOO MANY FILES (STAT)

STAT displays this message if there is insufficient memory available to sort and display all of the files on the specified disk. Try limiting the number of files it has to sort by judicious use of ambiguous file names.

UNRECOGNIZED DESTINATION (PIP)

PIP displays this message if you specify an "illegal" destination device.

VERIFY ERROR (PIP)

If you use the "[v]" (verify) option of PIP when copying to a disk file, PIP will write a sector to the disk, read it back, and compare the data. PIP displays this message if the data does not match.

If there is a problem with your disk system, you should have seen some form of disk error message preceding this one. If there is no preceding message, then you have a problem with the main memory on your system.

Wrong CP/M Version (Requires 2.0) (STAT)

Self-explanatory.

(XSUB ACTIVE) (XSUB)

This is not really an error message, but you may mistake it for one. XSUB is the eXtended SUBMIT program. Without it, SUBMIT can only feed command lines to the Console Command Processor. XSUB allows character-by-character input into any program that uses the BDOS to read console input.

XSUB is initiated by being the first command in a SUB file. Once initiated it stays in memory until the end of the SUB file has been reached. Until that happens, XSUB will output this message every time a warm boot occurs as a reminder that it is still in memory.

XSUB Already Present (XSUB)

XSUB will display this message if it is already active and you attempt to load it again.

Miscellaneous Errors

This section deals with errors that are not accompanied by any error message. It is included here to help you recognize a problem after it has already occurred. The errors are shown grouped by product.

ASM: Fails to Detect Unterminated IF Clause

If you use the IF pseudo-operation, it must be followed by a matching ENDIF. ASM fails to detect the case that the end of the source file is encountered *before* the ENDIF.

If the condition specified on the IF line is false, you could have a situation in which ASM would ignore the majority of the source file without comment.

ASM: Creates HEX File That Cannot Be Loaded

If you omit the ORG statement at the front of a source file, ASM will assemble the code origined at location 0000H. This file will crash the system if you try to load it with DDT. The message "ERROR: INVERTED ADDRESS" will be shown from LOAD.

CP/M: Signs On and Then Dies Without A > Prompt

After the BIOS has signed on, it transfers control to the Console Command Processor. The CCP then attempts to log in the system disk, reading the file directory and building the allocation vector. If your file directory has been badly corrupted, it can cause the system to crash. Use another system disk and try to display the directory on the bad disk.

DDT: Loads HEX File and Then Crashes the System

DDT does not check the addresses specified in a HEX file. If you have forgotten to put an ORG statement at the front of the source file, or more subtly, if your source program has "wrapped around" by having addresses up at 0FFFFH and "above," the assembler will start assembling at 0000H again.

DIR: Shows Odd-Looking File Names

If you have odd-looking file names, or the vertical lines of ":" that DIR uses to separate the file names are misaligned, then the file directory has been corrupted. One strategy is to format a new disk, copy all of the valid files to it, and discard the corrupted disk.

DIR: Shows More than One Entry with the Same Name

This can happen if you use a program that creates a new file without asking the BDOS to delete any existing files of the same name. It can also happen if you use the custom MOVE utility carelessly.

To remedy the situation proceed as follows:

- Use PIP to copy the specific file to another disk. Do not use an ambiguous file name; specify the duplicated file name exactly. PIP will copy the first instance of the file it encounters in the directory.
- Use the ERA command to erase the duplicated file. This will erase both copies of the file.
- Use PIP to copy back the first instance of the file.

STAT: User Numbers > 15

If you use the "STAT USR:" command to display which user numbers contain active files, and user numbers greater than 15 are displayed, then the file directory on the disk has been corrupted.

Use PIP to copy the valid files from legitimate user numbers, and then discard the corrupted disk.

SUBMIT: Fails to Start Submit Procedure

There are several reasons why SUBMIT will not initiate a SUB file:

• You are using the standard release version of SUBMIT and your current default disk is other than drive A:. SUBMIT builds its "\$\$\$.SUB" file on the default disk, but the CCP only looks on drive A: for "\$\$\$.SUB". Use the following procedure to modify SUBMIT to build its "\$\$\$.SUB" file on drive A:

```
A><u>DDT SUBMIT.COM<cr></u>

DDT VERS 2.2

NEXT PC

0600 0100

-<u>55bb</u> <- Change 5bb

05BB 01 <u>00<cr></u> <- from 00 (default drive)

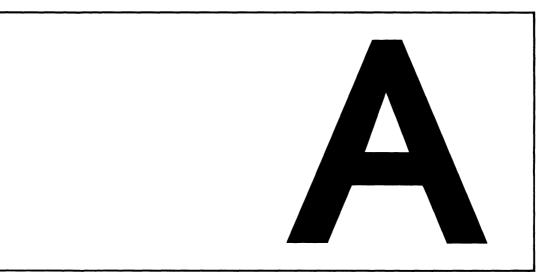
05BC 24 <u>.<cr></u> to 01 (drive A:)

-<u>^C</u>

A><u>SAVE 5 SUBMIT.COM<cr></u>

A>_
```

- If you forgot to terminate the last line of the SUB file with a CARRIAGE RETURN.
- If your SUB file contains a line with nothing but a CARRIAGE RETURN on it (that is, a blank line).



ASCII Character Set

The American Standard Code for Information Interchange (ASCII) consists of a set of 96 displayable characters and 32 nondisplayed characters. Most CP/M systems use at least a subset of the ASCII character set. When CP/M stores characters on a diskette as text, the ASCII definitions are used.

Several of the CP/M utility programs use the ASCII Character Code. Text created using ED is stored as ASCII characters on diskette. DDT, when displaying a "dump" of the contents of memory, displays both the hexadecimal and ASCII representations of memory's contents.

ASCII does not use an entire byte of information to represent a character. ASCII is a seven-bit code, and the eighth bit is often used for *parity*. Parity is an error-checking method which assures that the character received is the one transmitted. Many microcomputers and microcomputer devices ignore the *parity bit*, while others require one of the following two forms of parity:

Even Parity

The number of binary 1's in a byte is always an even number. If there is an odd number of 1's in the character, the parity bit will be a 1; if there is an even number of 1's in the character, the parity bit is made a 0.

Odd Parity

The number of binary 1's in a byte is always an odd number. If there is an

even number of 1's in the character, the parity bit will be a 1; if there is an odd number of 1's in the character, the parity bit is made a 0.

Alternative ways of *coding* the information stored by the computer include the 8-bit EBCDIC (Extended Binary Coded Decimal Interchange Code), used by IBM, and a number of *packed binary* schemes, primarily used to represent numerical information.

				$b7 \longrightarrow b6 \longrightarrow b5 \longrightarrow$	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	Col.	0 1	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	Р	`	р
ŏ	ŏ	0	1	1	SOH	DC1	!	1	Ă	Q	а	q
Ō	ō	1	0	2	STX	DC2	"	2	В	R	ь	r
ō	0	1	1	3	ETX	DC3	#	3	C	S	с	s
Ō	1	0	0	4	EOT	DC4	\$	4	D	Т	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK		&	6	F	V	f	v
0	i	1	1	7	BEL	ETB	,	7	G	w	g	w
1	0	0	0	8	BS	CAN	(8	Н	X	ĥ	x
1	0	0	1	9	НТ	EM)	9	I	Y	i	у
1	0	1	0	10	LF	SUB	*	:	J	Z	l j	z
1	0	1	1	11	VT	ESC	+	;	K		k	{
1	1	0	0	12	FF	FS	,	<	L	Ň	1	
1	1	0	1	13	CR	GS	-	=	М]	m	
1	1	1	0	14	so	RS		>	Ν	^	n	~
1	1	1	1	15	SI	US	/	?	0	-	0	DEL
NUL Null					DCI	Dev	vice cor	ntrol 1				
SOH			ading				DC2	Dev	vice cor	ntrol 2		
SOH Start of heading STX Start of text					DC3	Dev	vice cor	ntrol 3				
ETX		of tex					DC4	Dev	vice cor	ntrol 4		
EOT	End	of tra	nsmiss	ion			NAK	Neg	gative a	cknow	ledge	
ENQ		uiry					SYN	Syn	chrono	ous idle	e	
ACK						ETB	Enc	l of tra	nsmiss	ion blo	ock	
BEL Bell or alarm					CAN	Car	ncel					
BS Backspace					EM	Enc	l of me	dium				
					SUB	Sut	ostitute					
LF Line feed					ESC	Esc						
VT						FS		e separa				
FF	F Form feed					GS	Gro	oup sep	arator			
CR						RS		ord sep		r		
SO	0					US	Uni	t separ	ator			
SI	Shif	ît in					SP	Spa	ice			
DLE	Dat	a link (escape				DEL	Del	ete			

Table A-1. ASCII Character Codes

•

Hexadecimal	Binary	ASCII	Hexadecimal	Binary	ASCII
00	000 0000	NUL	30	011 0000	0
01	000 0001	SOH	31	011 0001	1
02	000 0010	STX	32	011 0010	2
03	000 0011	ETX	33	011 0011	3
04	000 0100	EOT	34	011 0100	4
05	000 0101	ENQ	35	011 0101	5
06	000 0110	ACK	36	011 0110	6
07	000 0111	BEL	37	011 0111	7
08	000 1000	BS	38	011 1000	8
09	000 1001	HT	39	011 1001	9
0A	000 1010	LF	3A	011 1010	:
0B	000 1011	VT	3B	011 1011	
0C	000 1100	FF	3C	011 1100	, <
0D	000 1101	CR	3D	011 1101	; < = > ?
0E	000 1110	SO	3E	011 1110	>
0F	000 1111	SI	3F	011 1111	2
			-		•
10	001 0000	DLE	40	100 0000	
11	001 0001	DC1	41	100 0001	A
12	001 0010	DC2	42	100 0010	B
13	001 0011	DC3	43	100 0011	С
14	001 0100	DC4	44	100 0100	D
15	001 0101	NAK	45	100 0101	E
16	001 0110	SYN	46	100 0110	F
17	001 0111	ETB	47	100 0111	G
18	001 1000	CAN	48	100 1000	Н
19	001 1001	EM	49	100 1001	I
1A	001 1010	SUB	4A	100 1010	J
1 B	001 1011	ESC	4B	100 1011	K
1C	001 1100	FS	4C	100 1100	L
1D	001 1101	GS	4D	100 1101	М
1E	001 1110	RS	4E	100 1110	Ν
1F	001 1111	US	4F	100 1111	0
20	010 0000	SP	50	101 0000	Р
21	010 0001	!	51	101 0001	Q
22	010 0010	"	52	101 0010	Ř
23	010 0011	#	53	101 0010	S
24	010 0100	\$	54	101 0100	Ť
25	010 0101	%	55	101 0100	Ū
26	010 0101	% &	56	101 0110	v
20	010 0111	æ,	57	101 0111	ŵ
28	010 1000	(58	101 1000	x
28	010 1000)	58 59	101 1000	Ŷ
29 2A	010 1010	, •	59 5A		Z
2A 2B	010 1010	+	5B	101 1010	[
2B 2C				101 1011	L \
2C 2D	010 1100	,	5C	101 1100	ì
	010 1101	-	5D	101 1101	, 1
2E 2F	010 1110	;	5E 5F	101 1110	
<u>۲</u>	010 1111	/	<u>ЭГ</u>	101 1111	-

 Table A-2.
 ASCII Character Codes in Ascending Order

Hexadecimal	Binary	ASCII	Hexadecimal	Binary	ASCII
60	110 0000		70	111 0000	р
61	110 0001	а	71	111 0001	q
62	110 0010	b	72	111 0010	r
63	110 0011	с	73	111 0011	s
64	110 0100	d	74	111 0100	t
65	110 0101	e	75	111 0101	u
66	110 0110	f	76	111 0110	v
67	110 0111	g	77	111 0111	w
68	110 1000	ĥ	78	111 1000	х
69	110 1001	i	79	111 1001	У
6A	110 1010	i	7A	111 1010	Z
6B	110 1011	k	7B	111 1011	{
6C	110 1100	1	7C	111 1100	Ĩ
6D	110 1101	m	7D	111 1101	}
6E	110 1110	n	7E	111 1110	~
6F	110 1111	0	7F	111 1111	DEL

`

 Table A-2.
 ASCII Character Codes in Ascending Order (Continued)

CP/M Command Summary

This appendix summarizes the command line format and the function of each CP/M built-in and transient command. The commands are listed in alphabetical order.

ASM Command Lines

- **ASM filename** <**cr**> Assembles the file filename. ASM; uses the currently logged disk for all files.
- ASM filename.opt<cr>
 Assembles the file filename.ASM on drive o: (A:,B:,...,P:). Writes HEX file on drive p: (A:,B:,...,P:), or skips if p: is Z:. Writes PRN file on drive t: (A:,B:,...,P:), sends to console if p: is X:, or skips if p: is Z:.

DDT Command Lines

- **DDT<Cr>** Loads DDT and waits for DDT commands.
- **DDT x:filename.typ**<**cr**> Loads DDT into memory and also loads filename.typ from drive x: into memory for examination, modification, or execution.

.

DDI Command Summary

Assss	Enters assembly language statements beginning at hexadecimal address ssss.		
D	Displays the contents of the next 192 bytes of memory.		
Dssss,ffff	Displays the contents of memory starting at hexadecimal address ssss and finishing at hexadecimal address ffff.		
Fssss,ffff,cc	Fills memory with the 8-bit hexadecimal constant cc starting at hexadecimal address ssss and finishing with hexadecimal address ffff.		
G	Begins execution at the address contained in the program counter.		
G,bbbb	Sets a breakpoint at hexadecimal address bbbb, then begins execution at the address contained in the program counter.		
G,bbbb,cccc	Sets breakpoints at hexadecimal addresses bbbb and cccc, then begins execution at the address contained in the program counter.		
Gssss	Begins execution at hexadecimal address ssss.		
Gssss,bbbb	Sets a breakpoint at hexadecimal address bbbb, then begins execution at hexadecimal address ssss.		
Нх,у	Hexadecimal sum and difference of x and y.		
lfilename.typ	Sets up the default file control block using the name filename.typ.		
L	Lists the next eleven lines of assembly language program disassembled from memory.		
Lssss	Lists eleven lines of assembly language program disassembled from memory starting at hexadecimal address ssss.		
Lssss,ffff	Lists the assembly language program disassembled from memory starting at hexadecimal address ssss and finishing at hexadecimal address ffff.		

Mssss,ffff,ddc	d Moves the contents of the memory block starting at hexadecimal address ssss and ending at hexadecimal address ffff to the block of memory starting at hexadecimal address dddd.
R	Reads a file from disk into memory (use "I" command first).
Rnnnn	Reads a file from disk into memory beginning at the hexadecimal address nnnn higher than normal (use "I" command first).
Sssss	Displays the contents of memory at hexadecimal address ssss and optionally changes the contents.
Innnn	Traces the execution of (hexadecimal) nnnn program instructions.
Unnnn	Executes (hexadecimal) nnnn program instructions, then stops and displays the CPU register's contents.
x	Displays the CPU register's contents.
Xr	Displays the contents of CPU or Flag r and optionally changes them.

DIR Command Lines

- **DIR x: Cr** Displays directory of all files on drive x:. Drive x: is optional; if omitted, the currently logged drive is used.
- **DIR x:filename.typ <cr>** Displays directory of all files on drive x: whose names match the ambiguous or unambiguous filename.typ. Drive x: is optional; if omitted, the currently logged drive is used.

DUMP Command Line

DUMP x:filename.typ <cr> Displays the hexadecimal representations of each byte stored in the file filename.typ on drive x:. If filename.typ is ambiguous, displays the first file which matches the ambiguous file name.

ED Command Line

ED x:filename.typ <cr> Invokes the editor, which then searches for filename.typ on drive x: and creates a temporary file x:filename.\$\$\$ to store the edited text. The filename.typ is unambiguous. Drive x: is optional; if omitted, the currently logged drive is assumed.

ED Command Summary

NOTE: Non-alphabetic commands follow the "Z" command.

nA	Append lines. Moves "n" lines from original file to edit buffer. 0A moves lines until edit buffer is at least half full.		
+/- B	Begin/Bottom. Moves CP. +B moves CP to beginning of edit buffer -B moves CP to end of edit buffer.		
+/- nC	Move by characters. Moves CP by "n" character positions. + moves forward - moves backward.		
+/—nD	Delete characters. Deletes "n" characters before or after the CP in the edit buffer. + deletes before the CP - deletes after the CP.		
E	End. Ends edit, closes files, and returns to CP/M; normal end.		
nFstring^Z	Find string. Finds the "n"th occurrence of string, beginning the search after the CP.		
н	Move to head of edited file. Ends edit, renames files, and then edits former temporary file.		
l <cr></cr>	Enter insert mode. Text from keyboard goes into edit buffer after the CP; exit with CONTROL-Z.		
Istring^Z	Insert string. Inserts string in edit buffer after the CP.		
lstring <cr></cr>	Insert line. Inserts string and CRLF in the edit buffer after the CP.		
nJfindstring^Z	Linsertstring ^ Zendstring ^ Z Juxtaposition. Beginning after the CP, finds findstring, inserts insertstring after it, then deletes all following characters up to but not including endstring; repeats until performed "n" times.		
+/—nK	Kill lines. Deletes "n" lines. + deletes after the CP - deletes before the CP.		
+/—nL	Move by lines. Moves the CP to the beginning of the line it is in, then moves the CP "n" lines forward or backward. + moves forward - moves backward.		

-

nMcommandstring^AZ Macro command. Repeats execution of the ED commands in

commandstring "n" times. "n" = 0, "n" = 1, or "n" absent repeats execution until error occurs.

- **nNstring^Z** Find string with autoscan. Finds the "n"th occurrence of string, automatically appending from original file and writing to temporary file as necessary.
- Return to original file. Empties edit buffer, empties temporary file, returns to beginning of original file, ignores previous ED commands.
- +/-**nP** Move CP and print pages. Moves the CP forward or backward one page, then displays the page following the CP. "nP" displays "n" pages, pausing after each.
- Quit edit. Erases temporary file and block move file, if any, and returns to CP/M; original file is not changed.
- **R**<**cr**> Read block move file. Copies the entire block move file X\$\$\$\$\$.LIB from disk and inserts it in the edit buffer after the CP.
- **Rfilename** < cr> Read library file. Copies the entire file filename with extension LIB from the disk and inserts it in the edit buffer after the CP.
- **nSfindstring^Zreplacestring^Z** Substitute string. Starting at the CP, repeats "n" times: finds findstring and replaces it with replacestring.

+/—nī	Type lines. Displays "n" lines.
	+ displays the "n" lines after the CP
	- displays the "n" lines before the CP.
	If the CP is not at the beginning of a line
	OT disates for a distingtion of the line of

- 0T displays from the beginning of the line to the CP T displays from the CP to the end of the line 0TT displays the entire line without moving the CP.
- +/-U Uppercase translation. After +U command, alphabetic input to the edit buffer is translated from lowercase to uppercase; after -U, no translation occurs.
- **OV** Edit buffer free space/size. Displays the decimal number of free (empty) bytes in the edit buffer and the total size of the edit buffer.
- +/-V Verify line numbers. After +V, a line number is displayed with each line displayed; ED's prompt is then preceded by the number of the line containing the CP. After -V, line numbers are not displayed, and ED's prompt is "*".

nW	Write lines. Writes first "n" lines from the edit buffer to the temporary file; deletes these lines from the edit buffer.
nX	Block transfer (Xfer). Copies the "n" lines following the CP from the edit buffer to the temporary block move file X\$\$\$\$\$.LIB; adds to previous contents of that file.
nZ	Sleep. Delays execution of the command which follows it. Larger "n" gives longer delay, smaller "n" gives shorter delay.
n:	Move CP to line number "n." Moves the CP to the beginning of the line number "n" (see " $+/-V$ ").
:m	Continue through line number "m." A command prefix which gives the ending point for the command which follows it. The beginning point is the location of the CP (see " $+/-V$ ").
+/ n	Move and display one line. Abbreviated form of $+/-nLT$.

ERA Command Lines

- **ERA x:filename.typ**<**cr**> Erases the file filename.typ on the disk in drive x:. The filename and/or typ can be ambiguous. Drive x: is optional; if omitted, the currently logged drive is used.
- **ERA x:*.*<Cr>** Erases all files on the disk in drive x:. Drive x: is optional; if omitted, the currently logged drive is used.

Line Editing Commands

- **CONTROL-C** Restarts CP/M if it is the first character in command line. Called *warm start*.
- **CONTROL-E** Moves to the beginning of next line. Used for typing long commands.
- **CONTROL-H or BACKSPACE** Deletes one character and erases it from the screen (CP/M version 2.0 and newer).
- **CONTROL-J or LINE FEED** Same as CARRIAGE RETURN (CP/M version 2.0 and newer).
- **CONTROL-M** Same as CARRIAGE RETURN (<cr>).
- **CONTROL-P** Turns on the list device (usually your printer). Type it again to turn off the list device.

- **CONTROL-R** Repeats current command line (useful with version 1.4); it verifies the line is corrected after you delete several characters (CP/M version 1.4 and newer).
- **CONTROL-S** Temporarily stops display of data on the console. Press any key to continue.

CONTROL-U or CONTROL-X Cancels current command line (CP/M version 1.4 and newer).

RUBOUT (RUB) or DELETE (DEL) Deletes one character and echoes (repeats) it.

Load Command Line

LOAD x:filename<**cr**> Reads the file filename.HEX on drive x: and creates the executable program file filename.COM on drive x:.

MOVCPM Command Lines

- **MOVCPM**<**cr**> Prepares a new copy of CP/M which uses all of memory; gives control to the new CP/M, but does not save it on disk.
- **MOVCPM nn < cr>** Prepares a new copy of CP/M which uses "nn" K bytes of memory; gives control to the new CP/M, but does not save it on disk.
- **MOVCPM * * <cr>** Prepares a new copy of CP/M that uses all of memory, to be saved with SYSGEN or SAVE.
- MOVCPM nn * <cr>
 Prepares a new copy of CP/M that uses "nn" K bytes of memory, to be saved with SYSGEN or SAVE. The "nn" is an integer decimal number. It can be 16 through 64 for CP/M 1.3 or 1.4. For CP/M 2.0 and newer "nn" can be 20 through 64.

PIP Command Lines

- **PIP**<**cr**> Loads PIP into memory. PIP prompts for commands, executes them, then prompts again.
- **PIP pipcommandline** <**cr**> Loads PIP into memory. PIP executes the command pipcommandline, then exits to CP/M.

PIP Command Summary

x:new.typ=y:old.typ[p]<cr> Copies the file old.typ on drive y: to the file new.typ on drive x:, using parameters p.

x:new.typ=y:old1.typ[p],z:old2.typ[q] < cr> Creates a file new.typ on drive x: that

consists of the contents of file old 1.typ on drive y: using parameters p followed by the contents of file old2.typ on drive z: using parameters q.

-

x:filename.typ=dev:[p]<cr> Copies data from device dev: to the file filename.typ on drive x:.

dev:=x:filename.typ[p]<cr> Copies data from filename.typ on drive x: to device dev:.

dst:=src:[p]<cr> Copies data to device dst: from device src:.

PIP Parameter Summary

В	Specifies block mode transfer.
Dn	Deletes all characters after the "n"th column.
E	Echoes the copying to the console as it is being performed.
F	Removes form feed characters during transfer.
Gn	Directs PIP to copy a file from user area "n."
Н	Checks for proper Intel Hex File format.
Ι	Ignores any :00 records in Intel Hex File transfers.
L	Translates uppercase letters to lowercase.
Ν	Adds a line number to each line transferred.
0	Object file transfer (ignores end-of-file markers).
Pn	Issues page feed after every "n"th line.
Qs^Z	Specifies quit of copying after the string "s" is encountered.
R	Directs PIP to copy from a system file.
Ss^Z	Specifies start of copying after the string "s" is encountered.
Tn	Sets tab stops to every "n"th column.
U	Translates lowercase letters to uppercase.
V	Verifies copy by comparison after copy finished.
W	Directs PIP to copy onto an R/O file.
Z	Zeroes the "parity" bit on ASCII characters.

PIP Destination Devices

CON:	PUN:	LST:	Logical devices
TTY:	PTP:	LPT:	
CRT:	UP1:	UL1:	
UC1:	UP2:		Physical devices
OUT:	PRN:		Special PIP devices

PIP Source Devices

CON:	RDR:		Logical devices
TTY:	PTR:		
CRT:	UR1:		
UC1:	UR2:		Physical devices
NUL:	EOF:	INP:	Special PIP devices

REN Command Line

REN newname.typ=oldname.typ<cr> Finds the file oldname.typ and renames it newname.typ.

SAVE Command Line

SAVE nnn x:filename.typ<cr> Saves a portion of the Transient Program Area of memory in the file filename.typ on drive x: where nnn is a decimal number representing the number of pages of memory. Drive x: is the option drive specifier.

STAT Command Lines

- **STAT**<**cr**> Displays attributes and amount of free space for all diskette drives accessed since last warm or cold start.
- **STAT x: Cr** Displays amount of free space on the diskette in drive x:.
- **STAT x:filename.typ** <**cr**>(**CP/M 2.0 and newer**) Displays size and attributes of file(s) filename.typ on drive x:. filename.typ may be ambiguous. x: is optional; if omitted, currently logged drive is assumed.
- **STAT x:filename.typ \$atr<cr>** Assigns the attribute atr to the file(s) filename.typ on drive x:. File filename.typ may be ambiguous. Drive x: is optional; if omitted, currently logged drive is assumed.
- **STAT DEV:** <**cr**> Reports which physical devices are currently assigned to the four logical devices.
- **STAT VAL: Cr**> Reports the possible device assignments and partial STAT command line summary.
- **STAT log:=phy:<cr>** Assigns the physical device phy: to the logical device log: (may be more than one assignment on the line; each should be set off by a comma).
- **STAT USR:** <**cr**> (**CP/M 2.0 and newer**) Reports the current user number as well as all user numbers for which there are files on currently logged disks.

STAT x:DSK < cr> (CP/M 1.4 and newer) Assigns a temporary write-protect status to drive x:.

SUBMIT Command Lines

- **SUBMIT filename** <**cr**> Creates a file \$\$\$.SUB which contains the commands listed in filename.SUB; CP/M then executes commands from this file rather than the keyboard.
- **SUBMIT filename parameters** Creates a file \$\$\$.SUB which contains commands from the file filename.SUB; certain parts of the command lines in filename. SUB are replaced by parameters during creation of \$\$\$.SUB. CP/M then gets commands from this file rather than the keyboard.

SYSGEN Command Line

SYSGEN <**cr**> Loads the SYSGEN program to transfer CP/M from one diskette to another.

TYPE Command Line

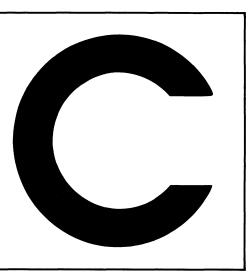
TYPE x:filename.typ<**cr**> Displays the contents of file filename.typ from drive x: on the console.

USER Command Line

USER n<cr> Sets the User Number to "n," where "n" is an integer decimal number from 0 to 15, inclusive.

x: Command Line

x:<cr> Changes the currently logged disk drive to drive x:. Drive x: can be "A" through "P."



Summary of BDOS Calls

Table C-1.	BDOS Function Definitions for CP/M-80 Version 2.2
------------	---

Function		Entry Exit		
No.	Name	Parameter(s)	Parameter(s)	Explanation
00	SYSTEM RESET	None	None	Restarts CP/M-80 by returning control to the the CCP after reinitializing the disk subsystem.
01	CONSOLE INPUT	None	A = ASCII character	Returns the next character typed to the character calling program.
				Any non-printable character is echoed to the screen (like BACKSPACE, TAB, or CARRIAGE RETURN). Execution does not return to the calling program until a character has been typed. Standard CCP control characters are recognized and their actions performed (CONTROL-P begins or ends printer echoing and so on).

Table C-1. (Continued)

Function		Entry	Exit	
No.	Name	Parameter(s)	Parameter(s)	Explanation
02	CONSOĒE OUTPUT	E = ASCII character	None	Displays the character in the E register on the console device. Standard CCP control characters are recognized and their actions performed (CONTROL-P begins or ends printer echoing and so on.).
03	READER INPUT	None	A = ASCII character	Returns the next character received from the reader device to the calling program. Execution does not return to the calling
04	PUNCH OUTPUT	E = ASCII character	None	program until a character is received. Transmits the character in the E register to the punch device.
05	LIST OUTPUT	E = ASCII character	None	Transmits the character in the E register to the list device.
06	DIRECT CONSOLE IN DIRECT CONSOLE OUT	E = FF hex E = ASCII character	A = ASCII None	If register E contains an FF hex, the console device is interrogated to see if a character is ready. If no character is ready, a 00 is returned to the calling program in register A otherwise the character detected is returned in register A. If register E contains any char- acter other than an FF hex, that character is passed to the console display. All CCP con- trol characters are ignored. The user must protect the program against nonsensical characters being sent from or received by the console device.
07	GET IOBYTE	None	A = IOBYTE	Places a copy of the byte stored at location 0003 hex in the A register before returning control to the calling program.
08	SET IOBYTE	E = IOBYTE	None	Places a copy of the value in register E into the memory location of 0003 hex before returning control to the calling program.
09	PRINT STRING	DE = String address	None	Sends the string of characters stored beginning at the address stored in the DE register pair to the console device. All characters in subsequent addresses are sent until BDOS encounters a memory location which contains a 24 hex (an ASCII "\$"). The CCP control characters are checked for and performed if encountered.
				ne H register in the A register if nothing is to be nufacturers, specifically Microsoft, make use of

such information to reduce movement of information between the H and A registers.

 Table C-1.
 (Continued)

Function		Entry	Exit	E
No.	Name	Parameter(s)	Parameter(s)	Explanation
0A	READ CONSOLE BUFFER	DE = Buffer address	Data in buffer	This function performs essentially the same as the CCP would in that it takes the characters the user types and stores them into the buffer that begins at the address stored in the DE register pair. The first byte in the buffer pointed to by the DE pair must be the maximum length of the command; BDOS will place the number of characters encountered in the second byte, with the typed command beginning with the third byte pointed to by the DE pair. All standard CCP editing characters are recognized during the command entry.
0B	GET CONSOLE STATUS	None	A = Status	BDOS checks the status of the console device and returns a 00 hex if no character is ready, FF hex if a character has been typed.
0C	GET VERSION NUMBER	None	HL = Version	If the byte returned in the H register is 00 hex then CP/M is present, if 01, then MP/M is present. The byte returned in the L register is 00 if the version is previous to CP/M 2.0, 20 hex if the version is 2.0, 21 hex if 2.1 and so on.
0D	RESET DISK SYSTEM	None		Used to tell CP/M to reset the disk subsystem. Should be used any time diskettes are changed.
0E	SELECT DISK	E = Disk number	None	Selects the disk to be used for subsequent disk operations. A 00 hex in the E register indicates disk A, a 01 hex indicates disk B, etc.
0F	OPEN FILE	DE = FCB address	A = 'Found'/ not found code	Used to activate a file on the current disk drive and current user area. BDOS scans the first 14 bytes of the designated FCB block and attempts to find a match to the filename in the block. A 3F hex (ASCII "?") can be used in any of the filename positions to indi- cate a "don't care" character. If a match is found, the relevant informa- tion about that file is filled into the rest of the FCB by CP/M-80. A value of 00 hex to 03 in register A upon return indicates the open operation was successful, while an FF hex indicates that the file could not be found. If question marks are used to identify a file, the first matching entry is used.
Note	: CP/ M-80 a	ways copies the	contents of th	tion about that file is fill the FCB by CP/M-80. A 03 in register A upon ret open operation was succ hex indicates that the fil If question marks are us the first matching entry e H register in the A regist

NOTE: CP/M-80 always copies the contents of the H register in the A register if nothing is to be specifically returned in the A register. Some manufacturers, specifically Microsoft, make use of such information to reduce movement of information between the H and A registers.

,

 Table C-1.
 (Continued)

Function		Entry	Exit	Evalenction	
No.	Name	Parameter(s)	Parameter(s)	Explanation	
10	CLOSE FILE	DE = FCB address	A = 'Found'/ not found code	Performs the opposite of the open file function. A close file function must be performed upon completion of use of any file which has had information written into it.	
11	SEARCH FOR FIRST	DE = FCB address	A = 'Found'/ not found code	Performs the same as the open file function with the difference being that the current disk buffer is filled with the 128-byte record which is the directory entry of the matched file.	
12	SEARCH FOR NEXT	None	A= 'Found'/ not found code	Performs the same as search for first function except that the search continues on from the last matched entry.	
13	DELETE FILE	DE = FCB address	A='Found'/ not found code	Changes a flag on the directory entry for the file pointed to by the FCB so that CP/M-80 no longer recognizes it as a valid file. No information is actually erased when this function is performed, although subsequent writes to diskette may use some of the area previously associated with the "deleted" file.	
14	READ SEQUEN- TIAL	DE = FCB address	A = Error code	If a file has been activated for use by an oper file or make file function, the read sequential function reads the next 128-byte block into memory at the current DMA address. The value of 00 hex is returned in the A register i the read was successful, while any nonzero value in the A register indicates failure.	
15	WRITE SEQUEN- TIAL	DE = FCB address	A = Error code	If a file has been activated for use by an open file or make file function, the write sequential function writes the 128-byte block of memory at the current DMA address to the next 128-byte record of the named file.	
16	MAKE FILE	DE = FCB address	A = DIR code	Creates a new file with the information (name) indicated by the FCB. CP/M-80 doe: not check to see if the file indicated already exists, so you must first check to see if the file exists (or delete it). A newly created file need not be opened, as the make file function also performs the necessary opening operations.	
17	RENAME FILE	DE = FCB address	A = DIR code	Changes the name of the file referenced by the first 16 bytes of the FCB to the name in the second 16 bytes.	

NOTE: CP/M-80 always copies the contents of the H register in the A register if nothing is to be specifically returned in the A register. Some manufacturers, specifically Microsoft, make use of such information to reduce movement of information between the H and A registers.

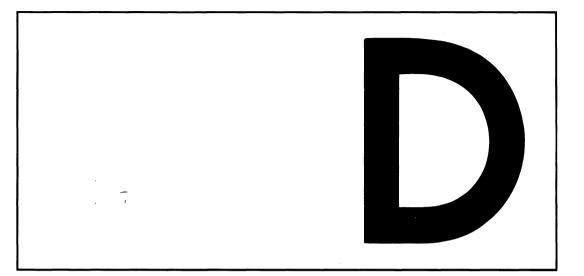
 Table C-1.
 (Continued)

unction	Entry	Exit	Explanation
Name	Parameter(s)	Parameter(s)	
RETURN LOGIN VECTOR	None	HL = Disk login	The bits in the HL register are used to specify which disk drives are active. The first bit in the L register refers to drive A, the last bit in the H register corresponds to drive P, the highest possible drive. A bit value of 1 indicates active status, a zero denotes an inactive drive.
RETURN CURRENT DISK	None	A = Current disk	The numbers 0 through 15 are used to represent the current default disk drive upon return from this function.
SET DMA ADDRESS	DE = DMA	None	Used to select the 128-byte memory block to be used for buffering all disk transfers. Upon system or disk reset, cold or warm start, the buffer is reset to 0080 hex on a normal CP/M -80 system.
GET ALLOC ADDRESS	None	HL = Alloca- tion address	Returns the starting address of the allocation vector, a table which is maintained in memory for each on-line disk drive that indi- cates the portions of the diskette which are in use.
WRITE Protect DISK	None	None	Provides temporary write protection for the diskette in the current default disk drive.
GET R/O VECTOR	None	$\begin{array}{l} HL = Disk \\ R/O \end{array}$	Returns a 16-bit value in the HL registers which indicate which drives on the system are write protected. The drives are assigned as in the LOGIN VECTOR, with a value 1 indicating write-protection.
SET FILE ATTRI- BUTES	DE = FCB address	A = DIR code	Sets the file attributes that indicate system/ directory and R/O or R/W file status for the file pointed to by the FCB address.
GET DISK PARMS	None	HL = DPB address	Retrieves the disk parameter block for the current active disk drive. These parameters can be used to determine space available on a diskette or to change the characteristics of the disk drive under user control.
GET USER CODE SET USER CODE	E = FF E = User code	A = Current User or None	If the E register contains an FF hex, the current user number is returned in the A reg- ister. To reset the user number, the appro- priate user code is placed in the E register. While the USER command allows user numbers in the range 0-15, this BDOS func- tion can set user numbers in the range of 0-31.
	Name RETURN LOGIN VECTOR RETURN CURRENT DISK SET DMA ADDRESS GET ALLOC ADDRESS WRITE PROTECT DISK GET R/O VECTOR SET FILE ATTRI- BUTES GET DISK PARMS GET USER CODE SET USER	NameParameter(s)NameParameter(s)RETURN LOGIN VECTORNoneRETURN CURRENT DISK SET DMA ADDRESSNoneGET ALLOC ADDRESSNoneGET ALLOC ADDRESSNoneWRITE PROTECT DISK GET R/O VECTORNoneSET FILE ATTRI- BUTES GET DISK GET DISK NoneDE = FCB addressGET USER CODE SET USERE = FF E = User	NameParameter(s)Parameter(s)Parameter(s)Parameter(s)RETURN LOGIN VECTORNoneHL = Disk loginRETURN CURRENT DISK SET DMA ADDRESSNoneA = Current diskGET ALLOC ADDRESSNoneHL = Alloca- tion addressGET ALLOC ADDRESSNoneHL = Alloca- tion addressWRITE PROTECT DISK GET R/O VECTORNoneNoneSET FILE BUTES GET DISK GET DISK BUTESDE = FCB addressA = DIR codeGET USER CODE SET USERE = FF E = UserA = Current User or None

1

Table C-1. (Continued)

Function		Entry	Exit	Explanation		
No.	Name	Parameter(s)	Parameter(s)	Dapianation		
21	READ RANDOM	DE = FCB address	A = Error code	Reads the random record number contained in the 33rd, 34th, and 35th byte (a 24-bit address) of the FCB pointed to.		
22	WRITE RANDOM	DE = FCB address	A = Error code	Writes information from the current DMA address to the random record pointed to by the number contained in the 33rd, 34th, and 35th bytes of the indicated FCB.		
23	COMPUTE FILE SIZE	DE = FCB address	RRF set	Returns the current size of the random record file in the three bytes that constitute the random record field of the FCB. If the third byte contains a 1, then the file contains the maximum record count of 65536, other- wise the value in the first two bytes is a 16-bit value that represents the file size.		
24	SET RANDOM RECORD	DE = FCB address	RRF set	Returns the next random record (fills in the random record field of the FCB) after the last sequentially read record. Digital Research suggests that this function is most appropriate to file indexing.		
25	RESET DRIVE	DE = Reset drive bits	A = Error code	Forces the specified drives to be reset to the drive bits initial non-logged status.		
28	WRITE RANDOM (ZERO)	DE = FCB address	A = Error code	Writes a record of all zeros to diskette before a record is written; useful for identifying unused random records (an unused record would contain zeros instead of data).		
specif	NOTE: CP/M-80 always copies the contents of the H register in the A register if nothing is to be specifically returned in the A register. Some manufacturers, specifically Microsoft, make use of such information to reduce movement of information between the H and A registers.					



Summary of BIOS Calls

Label in Jump Table	Entry Parameter(s)	Exit Parameter(s)	Explanation
COLDSTART	None 1	C = 0	Your routine should perform all the necessary start-up operations, including initializing all the values in the base page. Before exiting, the C register must be set to zero.
WARMSTART	None	C = Drive	Your routine should perform all the necessary restart operations but does not need to reinitialize the base page. The C register, on exit, should contain the cur- rent drive number.
CONSOLE STATUS (CONST)	None	A = Status	
CONSOLE* INPUT	None	A = Character	

Table D-1. (Continued)

Label in Jump Table	Entry Parameter(s)	Exit Parameter(s)	Explanation
READER* INPUT	None	A = Character	Your routine should wait for a character to be entered at the appropriate device and then return the character in the A register.
CONSOLE* OUTPUT	C = Character	None	· · · · · · · · · · · · · · · · · · ·
LIST* OUTPUT	C = Character	None	
PUNCH* OUTPUT	C = Character	None	Your routine should take the character in the C register and display it on the appropriate device.
HOME DISK	None	None	The head of the disk drive should be returned to the home position (track 0, sector 0).
SELECT DISK	C = Drive	HL = DHA	Your routine should select the drive indi- cated by the number in the C register. The HL register on return should contain the address of the disk parameter header.
SET TRACK	C = Track	None	The track indicated by the C register value should be set as the next track to be accessed by the disk drive.
SET SECTOR	C = Sector	None	The sector indicated by the C register value should be set as the next track to be accessed by the disk drive.
SET DMA ADDRESS	BC = DMA address	None	The DMA address indicated by the BC register pair should be set as the address to use for all information transfers from memory to diskette and vice versa.
READ DISK	None	A = Status	Read the current track and sector and transfer the data to the DMA address already set. A 01 hex should be returned if there was an error during transfer.
WRITE DISK	None	A = Status	Write the current track and sector from the data at the DMA address.
SECTOR	BC = Logical sector	HL = Physical sector	
TRANSLATION	DE = Sector map address		A special routine used for systems which maintain data in other than 128-byte blocks. The logical sector on entry is changed to reflect the appropriate actual sector on the diskette.
LIST STATUS	None	A = Status	Your routine should interrogate the appropriate device to see if a character is ready and return a 00 hex in the A regis- ter if not ready, or a FF hex if ready.

*All console and device I/O should be done by first looking at the IOBYTE (0003 hex) to determine which device is selected.

Index

A

ANSI Standard Escape sequences: Support via BIOS, 220 ASCII: Updating the time in ASCII, 224 ASM: Assembler, 185 Manual, 6 AUX: Logical Auxiliary (Reader/Punch) device, 56 Allocation block: Choosing size, 18 Concepts, 18 In file directory entry, 26 Maximum number in disk parameter block, 34 Prereading used block prior to writing, 155 Reserving in disk parameter block, 35 Allocation vector: Finding address of, 119 Pointer in disk parameter header, 32 Ambiguous file names: Avoidance in Rename File, 116 Concepts and restrictions, 24 Example processing, 401 Suggestion for utility program, 426 Used in BDOS Open File, 99 Used in DIR, 50 Used in ERA. 52 Used in Search for First Name Match, 103 Argc, argv: C Functions for command parameters, 405 Assign: C program, assigns logical to physical devices, 439 Attributes: In file directory entry, 26 Available RAM: Finding amount available, 65

B

BASIC: Problems with "gobbling" characters, 218 **BDOS:** Accessing file directory, C functions, 408 Entry Point, in base page, 59 Errors detected, 296

BDOS Function:

0, System Reset, 71 1, Read Console Byte, 72 2, Write Console Byte, 73 3, Read Reader Byte, 75 4, Write Punch Byte, 77 5, Write List Byte, 77 6, Direct Console I/O, 79 7, Get IOBYTE Setting, 80 8, Set IOBYTE, 86 9, Display \$-Terminated String, 88 10, Read Console String, 90 11, Read Console Status, 94 12, Get CP/M Version Number, 94 13, Reset Disk System, 95 14, Select Logical Disk, 97 15, Open File, 98 16, Close File, 102 17, Search for First Name Match, 103 18, Search for Next File Name Match, 107 19, Erase (Delete) File, 108 20, Read Sequential, 109 21, Write Sequential, 110 22, Create (Make) File, 112 23, Rename File, 115 24, Get Active Disks, 116 25, Get Current Default Disk, 118 26, Set DMA (Read/Write) Address, 118 27, Get allocation vector, 119 28, Set Logical Disk Read-Only, 120 29, Get Read-Only Disks, 120 30, Set File Attributes, 121 31, Get Disk Parameter Block Address, 125 32, Set/Get User Number, 131 33, Read Random, 131 34, Write Random, 133 35, Get File Size, 142 36, Set Random Record Number, 142 37, Reset Logical Disk Drive, 143 40, Write Random with Zero-fill, 144

BDOS Function codes: 69

In LIBRARY.H, 391 Initialization concepts, 12 Interface to other software, 15 Introduction to function calls, 20 Making a function request, 68 Making calls in C, 395

BDOS Function codes (continued)

Naming conventions, 68 Register conventions for function requests, 70 Use of Function 0 after hardware error, 299 Use of Function 0 after printer error, 224 Use of location 0005H, 14 What the BDOS does, 67

BDOS Error:

Bad Sector, 98, 154 R/O, 120 Select, 98, 153

BIOS:

Blocking/Deblocking, 152 Bootstrap functions, 148 CONIN, console input, 151 CONOUT, console output, 151 CONST, console input status, 150 Character drivers, debugging, 354 Components, 147 Configuration Block, accessing from C, 396 Debugging, 353 Debugging interrupts service routines, 357 Device table, accessing from C, 398 Different types of disk write, 155 Direct BIOS calls, example code, 156 Direct calls, examples, 65 Direct calls to read/write disk from C, 399 Disk Parameter Block, accessing from C, 398 Enhanced BIOS listing, 235 Enhanced data structures, 225 Enhancements, 209 Enhancements to support different protocols, 218 Entry points, 148 Example code for standard BIOS, 158 Feature checklist for debugging, 354 Finding the jump vector in RAM, 56 Function key table, accessing from C, 397 HOME disk heads, 153 Hardware error handling functions, 296 Host Buffer, HSTBUF, 152 Initialization concepts, 12 Interface to other software, 15 Jump numbers in LIBRARY.H, 391 Jump vector, 15, 56 Keeping the current date, 224 Keeping the current time, 224 LIST, list output, 151 LISTST, list device output status, 156 Live testing, 368 Logical Input/Output, 15 Making calls in C, 396

BIOS (continued) PUNCH (Auxiliary) output, 151 Preparing a special version, 184 **READ** sector, 154 **READER** input, 152 SECTRAN, logical to physical sector translation, 156 SELDSK, select disk, 153 SETDMA, set DMA address, 154 SETSEC, set sector, 153 SETTRK, set track, 153 Sequence of operations for sector write, 155 Support of function keys, 210 Using PIP to test, 369 WRITE sector, 155 What needs to be tested, 354 When to avoid direct calls, 15 **Backspace:** CONTROL-H. 47 **Bad sector management: 303** In the BIOS, 154 Suggestion for utility program, 426, 448 Base page: Current user number, 59 Example memory dumps, 61 Set by the CCP for loaded program, 54 **Basic Debugging for a BIOS: 320 Basic Disk Operating System:** See BDOS **Baud rates:** Speed, C program to set Baud rates, 431 **Bit Bucket:** If no Punch driver used, 77 Bit map: See Allocation vector Bit vector: As used in C functions, 402 Boolean AND, by_and, Code, 389, Narrative, 404 Definition of structure in LIBRARY.H, 395 Display, bv_disp, Code, 389, Narrative, 404 Fill, bv_fill, Code, 387, Narrative, 404 Inclusive OR, bv_or, Code, 389, Narrative, 404 Make, bv_make, Code, 387, Narrative, 404 Set bit, bv_set, Code, 387, Narrative, 404 Test bit, bv_test, Code, 388, Narrative, 404 Test bit non-zero, Code, 388, Narrative, 404 Block mask: In disk parameter block, 33

Block shift: In disk parameter block, 33 Blocking/Deblocking:

Concepts, 36

Blocking/Deblocking (continued)

Disk write types from BDOS to BIOS, 155 In the BIOS, 152

Bootstrap loader: Building a new version, 184 Debugging, 351 Example code, 197 Overview, 8

Buffer overflow: Debugging character driver, 358

Buffer thresholds: Debugging character driver, 359

Buffer wraparound: Debugging character driver, 360

Building a new CP/M system: Example console dialog, 206 The major steps, 183

Building an index file: Using Set Random Record Number, 143

Building your first CP/M system: 138

Built-in commands: In the CCP, 46

Built-in debug code: 321

Bv_and: Bit vector, boolean AND, Code, 389, Narrative, 404

Bv_disp: Bit vector, display, Code, 389, Narrative, 404

Bv__fill: Bit vector, fill, Code, 387, Narrative, 404

Bv_make: Bit vector, make, Code, 387, Narrative, 404

Bv_nz: Bit vector, test bit non-zero, Code, 388, Narrative, 404

Bv_or: Bit vector, inclusive OR, Code, 389, Narrative, 404

Bv_set: Bit vector, set bit, Code, 387, Narrative, 404

Bv_test: Bit vector, test bit, Code, 388, Narrative, 404

С

C Language: Reference manuals, 4 Use for utility programs, 371

C programs: ASSIGN, assigns logical to physical devices, 439 DATE, sets the date, 442 ERASE, a safer way to erase files, 409

C programs (continued)

FIND, finds lost files, 416 FUNKEY, sets the function keys, 445 MAKE, makes files visible/invisible, 427 MOVE, moves files between user numbers, 423 PROTOCOL, sets serial line protocols, 434 SPACE, shows used/free disk space, 420 SPEED, sets Baud rates, 431 TIME, sets the time, 442 UNERASE, restores erased files, 412 CBIOS.ASM: An ingredient for a new system, 185 CCP: Base page, set for program loaded, 185 Built-in commands, 50 Command Line Editing, 46 Control characters and their effects, 47 Default DMA buffer in base page, 61 Details, 45 ERA, erase (delete) files, 51 Example memory dumps of base page, 61 Functions, 46 Initialization concepts, 12 Interface to other software, 15 Logical devices, 56 Modifying the prompt to show the user number, 235 Overview, 12 Overwriting to gain memory, 45 Program loading, 54 Prompt, 46 REN, rename file, 52 Reloading on warm boot, 45 Resident commands, 14 Returning without warm boot, 66 SAVE, save memory image on disk, 53 Setting of command tail in base page, 60 Setting of default FCB's in base page, 60 TYPE, type an ASCII file, 52 USER, changing user number, 53 CCPM: Example of Get CP/M Version Number, 95

CDISK: Example of Reset Disk System, 96

COM file structure: 194

COM files: Loaded by the CCP, 46

CON:

Logical console, 16 CONIN:

Accessing the date and time, 223

CONIN (continued)

Console input, in the BIOS, 151 Recognizing incoming function key characters, 221 Use with forced input, 219

CONOUT:

Console output, in the BIOS, 151 Escape sequences to input date and time, 223 Processing output escape sequences, 222

CONST:

Console input status, in the BIOS, 50 Problems with programs that "gobble" characters, 218 Use with forced input, 219

CP/M: Bringing up a new system, 350

CP/M 128-byte "records": 41

CP/M file system: Concepts, 17

CP/M records as 128-byte sectors: 71

CRC: See Cyclic Redundancy Check

CRF: Example of Random Write, 135

Cancel command line: CONTROL-U, 49

Captions: For debug subroutines, 322

CARRIAGE RETURN: CONTROL-M, 48

Changed diskette: Size of buffer for detection, in disk parameter block, 36 Work area in disk parameter header, 32

Changing disks: Need to force disk log-in, 96

Changing user number: **USER**, 53

Character drivers: Example testbed, 355

Character I/O: Enhancements, 213 In the BIOS, 150 Interrupts for input, 215 Practical handling of errors, 299

Choosing allocation block size: 18

Circular buffer: For interrupt-driven input, 217 Structure in device table, 226

Close File: BDOS Function 16, 102

Code table:

Definition of structure in LIBRARY.H, 394 Display all strings, ct_disps, Code, 385, Narrative, 407 Get string for code, ct_strc, Code, 386, Narrative, 407 Get string for index, ct_stri, Code, 386, Narrative, 407 Initialize, ct_init, Code, 384, Narrative, 407 Prompt and return code, ct_parc, Code, 384, Narrative, 407 Return code, ct__code, Code, 385, Narrative, 407 Return index, ct_index, Code, 386, Narrative, 407 Used for command tail parameters, 406 **Cold Boot: BIOS functions**, 149 Concepts, 12 **Command line:** Canceling, CONTROL-U, 49 Deleting last character typed, 49 Repeating, CONTROL-R, 49 **Command Line Editing:** By the CCP, 46 Command tail: Code tables, C functions, 405 Example program to process parameters, 63 In base page, 60 Input to the CCP, 46 Processing, C functions, 405 **Communications:** Using Reader/Punch (Auxiliary), 151 Comp_fname: Compare file name, Code, 374, Narrative, 401 Compare file name: Comp_fname, Code, 374, Narrative, 401 **Configuration Block:** Accessing from C, 396 Concepts, 211 Suggestion for utility program, 448 Variable codes in LIBRARY.H. 391 **Console Command Processor:** See CCP

Console output: From debug subroutines, 323 Temporary pause, CONTROL-S, 47

Console output to printer: CONTROL-P, 48

Console status: Debugging character driver, 360

Control characters: Used in CCP command line editing, 47 Default disk: Changing, 50 In base page, 59 In CCP prompt, 46 **Default File Control Blocks:** In base page, 60 **Deferred writes:** In conjunction with track buffering, 231 **Delete character:** Rubout/Del, 49 **Deleting files:** ERA, 51 **Device table:** Accessing from C. 398 Displaying for debugging, 356 Structure, 225 **Digital Research:** Manuals, 6 **Direct BIOS calls:** Example code, 156 Examples, 65 When to avoid, 15 **Directory code:** As returned by BDOS calls, 71 As returned from Create (Make) File, 114 As returned from Rename File, 116 Returned by BDOS Close File, 103 Returned by BDOS Open File, 99 Returned by Search for First Name Match, 103 Returned by Search for Next Name Match, 107 **Directory entry: 99** Definition in LIBRARY.H, 394 **Directory Parameter Block:** Definition in LIBRARY.H, 393 **Disk Drivers:** Debugging, 364 Disk I/O: Enhancements, 231 In the BIOS, 152 Disk Map: In file directory entry, 26 **Disk Parameter Block:** Accessing from C, 398 Adding extra information, 41 Block shift, mask, and extent mask, 33 Definition in LIBRARY.H, 394 Details, 33 Finding the address of, 125 Maximum allocation block number, 34

Disk Parameter Block (continued) Number of directory entries -1, 35Number of tracks before directory, 36 Pointer in disk parameter header, 31 Reserving allocation blocks for file directory, 35 Sectors per track, 33 Size of buffer for detecting changed diskettes, 36 Worked example for hard disk, 39 **Disk Parameter Header:** Details, 28 Disk buffer, 31 Disk parameter block, 31 Pointer to allocation vector, 32 Sector skewing, 28 Work area for changed diskette detection, 32 Disk buffer: In disk parameter header, 31 Disk definition tables: Concept, 18 Details, 27 **Disk drivers:** Example testbed code, 365 **Disk errors:** Strategy, 303 Disk full: Error returned from Sequential Write, 112 **Disk layout:** CP/M on diskettes, 189 Disk map: As used in C functions, 402 Disk map clear: Dm_clr, Code, 382, Narrative, 403 Disk map display: Dm_disp, Code, 382, Narrative, 403 Diskette: Lavout of standard CP/M diskette, 37 **Diskette format:** Concepts, 9 **Display \$-Terminated String: BDOS Function 9, 88 Display directory error:** Err_dir, Code, 381, Narrative, 400 **Displaying an ASCII file: TYPE**, 52 **Displaying current user number: 54** Dm_clr:

Disk map clear, Code, 382, Narrative, 403 **Dm_disp:** Disk map display, Code, 382, Narrative, 403

Control characters (continued)

CONTROL-C: Used to abort after BDOS error, 98

CONTROL-P: Errors generated, 299

CONTROL-Z: If no Reader driver in BIOS, 75 Used to indicate end of file, 110 Used to terminate prior to BDOS Close File, 103

Conv_dfname: Convert directory file name, Code, 375, Narrative, 402

Conv_fname: Convert file name, Code, 375, Narrative, 408

Convert directory file name: Conv_dfname, Code, 375, Narrative, 402

Convert file name: Conv_fname, Code, 375, Narrative, 408

Create (Make) file: BDOS Function 22, 112

Ct_code: Code table, return code, Code, 385, Narrative, 407

Ct_disps: Code table, display all strings, Code, 385, Narrative, 407

Ct__index: Code table, return index, Code, 386, Narrative, 407

Ct_init: Code table, initialize, Code, 384, Narrative, 407

Ct_parc: Code table, prompt and return code, Code, 384, Narrative, 407

Ct_strc: Code table, get string for code, Code, 386, Narrative, 407

Ct_stri: Code table, get string for index, Code, 386, Narrative, 407

Current default drive: 97

Current logical disk: In base page, 59

Current record number: In FCB, unchanged for Random Read, 132 In FCB, unchanged for Random Write, 132

Current user number: Displaying, 54 In base page, 59

Customization: Of CP/M, an overview, 8 Cyclic Redundancy Check: As used in disk errors, 303

D

DDT:

Dynamic Debug Tool, 185, 329 Manual, 6 I Command used for building new CP/M system, 195 R Command used for building new CP/M system, 195 Used for checking CP/M images, 204 Used for debugging character drivers, 354 Used to create CP/M memory image, 194 Used to debug disk drivers, 364 **DESPOOL:** Use of LISTST BIOS entry, 156 **DIR:** Display directory of files, 50 **DMA buffer:** Default in base page, 60 **DPB:**

See Disk Parameter Block

See Disk Parameter Header

DTR: PROTOCOL, C program to set protocols, 434 See Data Terminal Ready

Data storage area: Concept, 17

Data Terminal Ready: Explanation of DTR protocol, 219

DATE: C program, sets the date, 442

Date: Keeping the current date in the BIOS, 224 Reading the date from the console driver, 223

Debug output: Controlling when it occurs, 324

Debug subroutines: 322 Overall design philosophy, 322

Debugging a new CP/M system, 319

Debugging checklist: Character output, 361 Disk drivers, 367 Interrupt service routines, 359 Non-interrupt service routine, 359 Real Time Clock, 362

Default DMA Address: 118 Default DMA buffer: In base page, 60

Index 493

DO: Suggestion for utility program, 448 DPB: See Disk Parameter Block DPH: See Disk Parameter Header

E

ED: Editor, manual, 6 ERA:

ERA: Erase (delete) files, 51

Echoing of keyboard characters: Read Console Byte, 72

End of File: Detection using Read Sequential, 110

Erase (Delete) File: BDOS Function 19, 108

ERASE: C program, a safer way to erase files, 409

Erased files: Unerasing them, 26

Erasing a file: ERA, 51 Logical deletion only, 23

Err_dir: Display directory error, Code, 381, Narrative, 400

Error messages: Debugging disk drivers, 368, Chapter 12

Errors: Dealing with hardware errors, 295 Example printer error routine, 301 Handling disk errors, 303 Hardware, analysis, 297 Hardware, correction, 299 Hardware, detection strategy, 296 Hardware, indication, 297 Improved disk error messages, 312 Practical handling, character I/O, 299

Escape sequences:

Function keys, debugging character driver, 360 Incoming, debugging character driver, 360 Processing output sequences, 222 Recognizing function key sequences, 222 Suggestion for utility program, 448 Support via device table, 226

Etx/Ack:

Debugging character drivers, 358, 362 Explanation of protocol, 219 Etx/Ack (continued) Protocol, C program to set protocols, 434

Example programs: Ordering diskette, 4

Extent: In file directory entry, 26 Of files, concepts, 18

Extent mask: In disk parameter block, 33

F

FCB: Default FCB's in base page, 60 See File Control Block FDOS: Rarely used term for BDOS/CCP combined

File Attributes: 99 Setting, 121 See File status

File Control Block:

Creating one from an ASCII file name, 100 Concepts, 18 Definition in LIBRARY.H, 393 Structure, 41 Used for random file operations, 43 Used for sequential file operations, 43 Used in BDOS Open File, 99 Used in BDOS Requests, 71

File Directory: Accessing entries directly, 399 Processing, C functions, 402

File Organizations: Concepts, 41

File Protection: Special characters in file name, 114 File changed:

File status bit in file directory entry, 26

File directory: Accessing, C functions, 400 Accessing, via BDOS & C functions, 408 Concept, 17 Details, 18 Disk map, 26 Displaying contents, DIR, 50 Entry structure, 22 Erasing files, ERA, 51

File extent, 26 File extent, 26 File name and type in entry, 27 Matching names, C functions, 401 Number of entries – 1, in disk parameter block, 35

File directory (continued)

Number of tracks before, 36 Record number, 27 Status (attribute) bits, 26 User number in entry, 22

File extent:

Concepts, 18 In file directory entry, 26 Manipulation to achieve Random I/O, 110-12 Opening extent 0 for Random I/O, 133-34

File name/type: In file directory entry, 23

File protection: Suggestion for utility program, 426

File status: In file directory entry, 26

File system: Concepts, 17

File type: Conventions for actual types, 24

Filecopy: Suggestion for utility program, 426

Files: Creating, sequence of operations, 20 Displaying a directory, DIR, 50

Find: C program, finds lost files, 416

Flushing buffers: Prior to BDOS Close File, 103

Forced input: Concepts, 219 Debugging character driver, 360 Suggestion for utility program, 448

Framing error: Character I/O, handling, 300

Function Key table: Accessing from C, 397

Function keys: Structure in LIBRARY.H, 392 Support with enhanced BIOS, 220 Testing in a live BIOS, 370

FUNKEY: C program, sets the function keys, 445

G

GETC: Example of Read Sequential, 111

GETDPB: Example of Get Disk Parameter Block Address, 126 GFA:

Example of Get File Attributes, 122 GNF: Example of Search First/Next File Name Match, 104

Get CP/M Version Number: BDOS Function 12, 94

Get Current Default Disk: BDOS Function 25, 118

Get Disk Parameter Block Address: BDOS Function 31, 125

Get Disk Parameter Block Address: Get_dpb, Code, 383

Get File Size: BDOS Function 35, 142

Get IOBYTE Setting: BDOS Function 7, 80

Get Read-Only Disks: BDOS Function 29, 120

Get allocation vector: BDOS Function 27, 119

Get configuration block address: Get_cba, 372

Get next directory entry: Get_nde, Code, 378, Narrative, 400

Get next file name: Get_nfn, Code, 376, Narrative, 408

Get_cba: Get configuration block address, 372

Get_dpb: Get Disk Parameter Block Address, Code, 383

Get_nde: Get next directory entry, Code, 378, Narrative, 400

Get_nfn: Get next file name, Code, 376, Narrative, 408

Η

HEX file structure: 195 HOME: Home disk heads, in the BIOS, 153 HSTBUF:

In the BIOS, 152 Hard disk:

Division into several logical disks, 39 Special considerations, 36

Hardware errors: Dealing with, 295, Chapter 9

Hardware reset: Debugging character driver, 359

Index 495

Heath/Zenith: Special version of CP/M, 55

Host Buffer: In the BIOS, 152

Host sector size: In the BIOS, 152

I/O Redirection: Assign, C program to assign physical devices, 439 Concepts, 214 IOBYTE Structure, 57

IF/ENDIF directives: Used for debug subroutines, 323

IOBYTE: Equates for bit fields, 86 Structure, 57 Use for polling communications line, 75 Use with Direct Console I/O for communications, 80

Initialization of debug subroutines: 323

Input redirection: Debugging character driver, 359

Input/Output: Fake I/O for debugging purposes, 327

Interactions: Between CCP, BDOS, and BIOS, 15

Interlace: See Sector skewing

Interrupt service routines: Debugging checklist, 357

Interrupts:

Architecture, 216 Circular buffers, 217 Dealing with buffer overflow, 219 Debugging service routines, 329 Use for character input drivers, 215

J

Johnson-Laird Inc.: Ordering diskette, 4 Jump vector: Use for entering the BIOS, 15

L

LIBRARY.C: Utility function library, 372 LIBRARY.H: Header for LIBRARY.C functions, 390 LIST: List output, in the BIOS, 151 LISTST: List device output status, in the BIOS, 156 LST: Logical list device, 56 Line editing: Using Read Console String, 91 Line feed: CONTROL-I. 48 List Device Errors: Problems with BDOS Function 5, 78 Loading CP/M: Overview, 11 Loading programs: Via the CCP, 54 Loadsvs: Suggestion for utility program, 448 Location 0000H: Use for warm boot, 13 Location 0005H: Simple examples of use, 20 Use for BDOS function calls, 14 Logging in a disk: Using BDOS Reset Disk System, 96 Logical deletion of files, 23 ERA, 51 Logical devices: CON:, LST:, AUX:, RDR:, PUN:, 56 Logical disk: As represented in File Control Block, 42 Division of hard disk into several logical disks, 39 Selecting, 97 Logical Input/Output: As afforded by the BIOS, 15 Logical records: Concepts, 41 Logical sectors to physical: 28 SECTRAN, in the BIOS, 156 Login Vector: See BDOS Function 24, 116 Lowercase letters in file name: 114 M-disk: Using memory as an ultra-fast disk, 232 M80: Macro Assembler, 185 MAC: Macro Assembler, 185 MAKE: C program, makes files visible/invisible, 427

MOVE: C program, moves files between user numbers, 423 MOVCPM: In conjunction with patches to CP/M, 234 Relocating the CCP and BDOS, 201 Use in building a new CP/M system, 182 MSGOUT: Example of Write Console Byte, 74 MSGOUTI:

Example of Write Console Byte, 74

Manuals: From Digital Research, 6

Maximum allocation block number: In disk parameter block, 34

Memory: Displaying in debug subroutines, 324 Finding size of area available for programs, 65 Use of hidden memory for buffers, 216 Used as an ultra-fast disk, 232

Memory dumps: Base page, 61

Memory image: Checking a new system, 204 Of new CP/M system, 185

Memory layout: For example BIOS, 190 For input to SYSGEN, 187 With CP/M loaded, 13

Messages: As an aid to debugging, 326

Ν

Notation: For example console dialog, 3 Number of file directory entries: In disk parameter block, 35

0

OM: Example of Display \$-Terminated String, 89 OPENF: Example of Open File, 100 Open File: BDOS Function 15, 98 Open directory: Open_dir, Code, 378, Narrative, 400 Open_dir: Open directory, Code, 378, Narrative, 400 Orville Wright approach to debugging: 320 Output Escape sequence: Debugging character output driver, 362 Overrun error: Character I/O, handling, 300 Overwriting the CCP: To gain memory, 45 Owner: Suggestion for utility program, 426

Ρ

PIP: Used to test a new BIOS, 369

PROM Bootstrap: Used to load CP/M, 11 PUN:

Logical Punch, 56 PUNCH:

Punch (Auxiliary) output, in the BIOS, 151 PUTC:

Example of Write Sequential, 113

PUTCPM: Example program, 191 Writing a utility, 189

Parallel printers: Error handling, 301

Parameters: Example program to process command tail, 63

Parity error: Character I/O, handling, 300

Pass counters: Use in debug subroutines, 324

Patching CP/M: General techniques, 234

Performance: Effect of sector skewing, 29

Physical end of line: CONTROL-E, 47

Physical sectors: Relative, on a hard disk, 38

Polled Reader Input: Problems and solutions, 75

Polled communications: Using Direct Console I/O, 80

Printer echo: CONTROL-P, 48

Printer errors: Example routine, 301 Use of watchdog timer, 224

Index 497

Printer timeout error: Handling, 300 **Program** loading: Via the CCP. 54 **Program termination:** Returning to CP/M, 66 Prompt: From the CCP, 46 **Protect/Unprotect:** Suggestion for utility program, 426 **PROTOCOL:** C program, sets serial line protocols, 434 **Protocol:** See also Data Terminal Ready, Request to Send, Xon/Xoff, Etx/Ack Definitions in LIBRARY.H. 392 Support in enhanced BIOS, 218 Support via device table, 226 Xon/Xoff, used by TYPE, 52 **Public files:** Patches to create this feature, 235

Suggestion for utility program, 448 **Public/Private:** Suggestion for utility program, 448

R

RAM-disk: Using memory as an ultra-fast disk, 232 RCS: Example of Direct Console I/O, 81 RDR: Logical Reader, 56 READ: Read Sector, in the BIOS, 154 **READER:** Reader input, in the BIOS, 152 **REN:** Rename file, 52 RF: Example of Rename File, 117 **RLSRDR:** Example of Read Reader Byte, 76 **RMAC:** Relocatable Macro Assembler, 185 RO: Example of Random File I/O, 136 RSA: Example of Read Console String, 92

RST7: Use for debugging drivers, 356

RTS: See also Buffer thresholds, Request to Send Protocol, C program to set protocols, 434

Random Read: Using Read Sequential, 110

Random Write: Using Write Sequential, 112

Random files: Concepts, 43 Creating an empty file, 144 Problem of sparse files, 44 Virtual size, 142

Random record number: In FCB, set for Random Read, 132 In FCB, set for Random Write, 132

Rd_disk: Read disk (via BIOS), Code, 377, Narrative, 400

Read Console Byte: BDOS Function 1, 72

Read Console Status: BDOS Function 11, 94

Read Console String: BDOS Function 10, 90

Read Random: BDOS Function 33, 131

Read Reader Byte: BDOS Function 3, 75

> Read Sequential: BDOS Function 20, 109

Read disk (via BIOS): Rd_disk, Code, 377, Narrative, 400

Read-Only: Automatic setting after changing diskettes, 32 File status bit in file directory entry, 26

Read-Only Disks: 120 Read-Only File: Attribute bit, 121

Read/write directory: Rw_dir, Code, 380, Narrative, 400

Reading/Writing disk: Direct BIOS calls from C, 399

Real Time Clock: Debugging, 362 Example testbed code, 363 TIME, C program to set the time, 444

Reclaim: Suggestion for utility program, 426

Record number: In file directory entry, 26 Manipulation to achieve Random I/O, 110, 112

Registers: Displaying in debug subroutines, 324

Relative page offset: Use for making direct BIOS calls, 65

Relative physical sectors: On a hard disk, 38

Release diskettes: Files from Digital Research, 6

Rename File: BDOS Function 23, 115

Renaming a file: REN, 52

Repeat command line: CONTROL-R, 48

Request to Send: Explanation of RTS protocol, 219

Reserved area: Concept, 17

Reset: Signal used to start loading of CP/M, 11

Reset Disk System: BDOS Function 13, 95

Reset Logical Disk Drive: BDOS Function 37, 143

Resident CCP commands: 14

Restoring registers: In interrupt service routine, 356

Rw_dir: Read/write directory, Code, 380, Narrative, 400

S

SAVE: Save memory image in disk file, 53 Use in building new CP/M system, 194

SECTRAN: Logical sector to physical, in the BIOS, 156 SELDSK:

Debugging disk drivers, 367 Select disk, in the BIOS, 153

SETDMA: Set DMA Address, in the BIOS, 154 SETSEC: Set Sector, in the BIOS, 153 SETTRK: Set Track, in the BIOS, 153 SETTRK/SEC: Debugging disk drivers, 367 SFA: Example of Set File Attributes, 122 SID: Debugging tool, 330 STAT: Use for displaying current user number, 54 SYSGEN: System Generator, 185 Writing a new system to disk, 186 Savesvs: Suggestion for utility program, 448 Saving memory on disk: SAVE, 53 Search First/Next: Example use together, 107 Search for file:

Srch_file, Code, 376, Narrative, 408 Search for Next File Name Match: BDOS Function 18, 107 Require for Search for First, 104

Sector interlace:

Sector skewing Sector size: Host, in the BIOS, 152

Sector skewing: Effect on performance, 29 For CP/M image on disk, 190 In disk parameter header, 28

Sector skipping: Concepts, 304

Sector sparing: Concepts, 304

Sectors: Use in allocation blocks, 18

Sectors per track: In disk parameter block, 33 Select Logical Disk:

BDOS Function 14, 97 Sequential Files:

Concepts, 43

Set DMA (Read/Write) Address: BDOS Function 26, 118

Required by Search for First Name Match, 104

Index 499

Set File Attributes: **BDOS Function 30, 121** Set IOBYTE: **BDOS Function 8, 86** Set Logical Disk Read-Only: BDOS Function 28, 120 Set Random Record Number: BDOS Function 36, 142 Set disk parameters for rd/wrt_disk: Set_disk, Code, 378, Narrative, 400 Set search control block: Setscb, Code, 381, Narrative, 401 Set/Get User Number: BDOS Function 32, 131 Set_disk: Set disk parameters for rd/wrt_disk, Code, 378, Narrative, 401 Setsch: Set search control block, Code, 381, Narrative, 401 Setterm: Suggestion for utility program, 448 Shadow PROM: Used to load CP/M, 11 Short: Minor change to C Language, 395 Single-density, single-sided: Diskette format, 10 Single disk reset, 143 Skewing: See Sector skewing Skipping: Skipping bad sectors on disk, 304 SPACE: C program, shows used/free disk space, 420 Spare: Suggestion for utility program, 448 Spare directory: Debugging disk drivers, 367 Sparing: Use of spare sectors on disk, 304 **Sparse Random Files:** Problem, 44 Special version of CP/M: Heath/Zenith, 55 SPEED: C program, sets baud rates, 431 Srch_file: Search for file, Code, 376, Narrative, 408

1

Sstrcmp: Substring compare, 373 Stack: Filling with known pattern, 323 Stack overflow: In interrupt service routine, 358 Standard BIOS: Example code, 158 String scan: Strscn, 372 String scan, uppercase: Ustrscn, 372 Strscn: String scan, 372 Structure: Of CP/M. 5 Subroutine: CCPM, Check if CP/M Version 2, 95 CDISK, Change Disk, 96 CRF, Create Random File, 135 DB\$Blank, Display a blank, 344 DB\$CAH, Convert A to ASCII Hex., 343 DB\$CRLF, Display Carriage Return, Line Feed, 344 DB\$Colon, Display a colon, 344 DB\$Conin, Debug console input, 336 DB\$Conout, Debug console output, 336 DB\$DAH, Display A in Hex., 343 DB\$DHLH, Display HL in Hex., 343 DB\$Display\$CALLA, Display call address, 343 DB\$Display, Main debug display, 338 DB\$GHV, Get Hex. Value, 348 DB\$Init, Debug initialize, 335 DB\$Input, Debug Port Input, 346 DB\$MEMORY, Debug display of memory/registers, 325 DB\$MSG, Display Message, 345 DB\$MSGI, Display Message (In-line), 345 DB\$Off, Turn debug output off, 337 DB\$On, Turn debug output on, 337 DB\$Output, Debug Port Output, 347 DB\$Pass, Decrement the pass counter, 337 DB\$Set\$Pass, Set pass counter, 337 DIVHL, Divide HL by DE, 129 FOLD, Fold lowercase to upper, 93 FSCMP, Folded String Compare, 93 GAB, Get Allocation Block given Track/Sector, 128 GDTAS, Get Directory Track/Sector, 127 GETC, Get Character from Sequential File, 111 GETDPB, Get Disk Parameter Block Address, 126 GFA, Get File Attributes, 122 GMTAS, Get Maximum Track/Sector, 127

Subroutine (continued)

GNF, Get Next File matching ambiguous name, 104 GNTAS, Get Next Track/Sector, 128 GTAS, Get Track/Sector from Allocation block No., 126 MSGOUT, Message Output, 74 MSGOUTI, Message Output In-Line, 74 MULHL, Multiply HL by DE, 129 OM, Output Message selected by A register, 89 OPENF, Open File given ASCII file name, 100 PUTC, Put Character to Sequential File, 113 RCS, Read Console String, 81 RF, Rename File, 117 RL\$RDR, Read Line from Reader, 76 RO, Random File I/O (non-128-byte records), 136 RSA, Return Subprocessor Address, 93 SDLR, Shift DE, HL one bit right, 141 SFA, Set File Attributes, 122 SHLR, Shift HL right one bit, 130 SUBHL, Subtract DE from HL, 130 **TERM**, Terminal Emulator, 87 TOUPPER, Fold lowercase to upper, 84 WL\$LST, Write Line to List Device, 79 WL\$PUN, Write Line to Punch, 78

Substring compare:

Sstrcmp, 373 Uppercase: Usstrcmp, 373

System file:

Attribute bit, 121 File status bit in file directory entry, 26 Not displayed by DIR, 51

System Reset: BDOS Function 0, 71

T

TERM: Example of Set/Get IOBYTE, 87

TIME: C program, sets the time, 442

TYPE: Type an ASCII file, 52

Tab:

Interaction of tab characters and escape sequences, 222

Tab expansion:

Supported by Write Console Byte, 73 Using Display \$-Terminated String, 89

Termination of programs, returning to CCP: 45 Testbed: Use for new drivers, 353

Time:

Correct display during debugging, 364 Keeping the current time in the BIOS, 224 Reading the time from the console driver, 223 Top of RAM: Finding, via base page, 60 **Track buffering:** Enhancement to disk I/O, 231 Track offset: See Tracks before directory Tracks before directory: In disk parameter block, 36 **Transient Program Area:** Finding available size, 65 Typeahead: Concepts, 217 Dealing with buffer overflow, 219

U

Undo command line: CONTROL-U, 49 UNERASE: C program, restores erased files, 412 User Number: Changing under program control, 131 Changing using USER, 53 Displaying, 54 In base page, 59 In file directory entry, 22 Patches to make this appear in CCP prompt, 235 Suggestion for utility program, 426 Usstrcmp: Uppercase substring compare, 373 Ustrcmp: Uppercase string scan, 372 Utility programs: 371

۷

Variable record lengths: Processing in Random Files, 133, 134

W

WL\$LST: Example of Write List Byte, 79 WL\$PUN: Example of Write Punch Byte, 78 WRITE: Write sector, in the BIOS, 155

Warm Boot: After BDOS Error, 98

Index 501

Warm Boot (continued)

BIOS functions, 150 Initiated by CONTROL-C, 47 Initiated by pressing a key, 94 Initiated by System Reset BDOS Function, 72 JMP at location 0000H, 55 Reloading the CCP, 45 Resetting Read-Only disks, 120 Setting default DMA Address, 118 Technique for avoiding, 66 Use of location 0000H, 13

Watchdog timer: Concepts, 225 Debugging Real Time Clock, 364 Use for detecting printer errors, 224

Write Console Byte: BDOS Function 2, 73

Write List Byte: BDOS Function 5, 77

Write Punch Byte: BDOS Function 4, 77

Write Random: BDOS Function 34, 133

Write Random with Zero-fill: BDOS Function 40, 144

-

Write Sequential: BDOS Function 21, 110 Write disk (via BIOS): Wrt_disk, Code, 377, Narrative, 400 Wrt_disk: Write disk (via BIOS), Code, 377, Narrative, 400

Χ

Xoff: CONTROL-S, 48 Xon: CONTROL-Q, 49 Xon/Xoff: Debugging character driver, 358, 362 Explanation of protocol, 240 PROTOCOL, C program to set protocols, 434 Supported by Read Console Byte, 72 Use by TYPE, 53

Z

į

ZSID: Z80 Symbolic Interactive Debugger, 185, 350